Compressed CNNs for Cover Song Identification

Vivian Li
vdl21

Abstract

With the internet continually evolving toward multimedia-based searches and information retrieval, audio content identification is becoming evermore important. Related to this problem is the challenge of cover song identification, seeking to identify similar songs despite considerable differences in tempo, pitch, and structure. In this paper, I train several deep convolutional neural networks to perform cover song identification, looking to optimise such models for mobile deployment. Evaluating these networks on datasets of classical and pop music, I look to investigate the effects of compression on classification accuracy and inference speed, informing the development of efficient and accurate music information retrieval models for mobile devices.

1 Introduction

Since the early 2000s, with the development of efficient music fingerprinting methods like Shazam [1], compact forms of music recognition have been an ongoing topic of research [2, 3, 4, 5]. However, such research has mostly been in the field of music fingerprinting, the process of matching an incoming sound *query* with a database of *reference* songs. However, fingerprinting techniques have struggled to generalise to more complicated forms of songs like classical music, where the musical interpretations of each performer introduce musical variations in tempo, expressive timings, instrumentation, and other musical aspects beyond the capabilities of most fingerprinting methods [6]. As such, the new task of cover song identification has emerged more recently, aiming to identify "cover songs," i.e. adaptations of an original song. This research also has broader applications beyond simple song identification, as the similarity metrics used in such research can also help recommendation algorithms find similar music (e.g., Spotify), or copyright detection software find infringing songs (e.g., YouTube Content ID).

In this paper, I implement multiple convolutional neural networks (CNNs) to detect cover songs and apply them to identify performances of classical pieces. Aiming to adapt cover song identification for mobile deployment, I test the impact of several compression methods, including pruning and quantisation, on the model's performance and inference speed. To assess possible applications of these CNNs for mobile and embedded devices, I further compare the reduced model sizes in the context of microprocessor memory requirements.

2 Related Work

The amount of research in cover song identification is extensive, with the best models competing in conferences like MIREX[1] Existing research has already obtained significant results through both deep learning and traditional approaches. Widmer et. al. [6] develop an adaptation of fingerprinting for classical music identification through an automatically compiled reference database. However, the use of fingerprinting leads to a large reference database, making this method unsuitable for mobile deployment. The system was also only tested on classical piano music. Rein et. al. [7] introduce a

[1] https://www.music-ir.org/mirex/wiki/MIREX_HOME

more traditional method of identifying classical music compositions using a combination of *wavelet dispersion vectors* as characteristic features and a simple two-layer neural net. In addition to this, more deep learning approaches have also been applied, with Siamese CNNs [8] used for live song identification [9] and CNNs used to extract latent representations for similarity comparison [10]. This work builds off of existing work on cover song identification using CNNs by Yu et. al. [11], aiming to adapt it for mobile deployment.

3 Background

3.1 Digital Signal Processing

The use of Fourier transforms is widespread throughout engineering and computer science. As the Fourier transform decomposes a time signal into its frequency components, the short-time Fourier transform can be used to track how those frequency components change over time, exposing important features like pitch and tempo for musical analysis.

However, the Fourier transform divides its output bins equally in frequency space. For musical applications, musical notes are situated logarithmically in frequency space, with one octave (7 notes) doubling in frequency. Consequently, the Fourier transform ends up under-sampling higher musical notes (Figure 1). Thus, musical applications often use the **constant-Q transform** (CQT), which adapts the short-time Fourier transform to be logarithmically spaced in frequency.

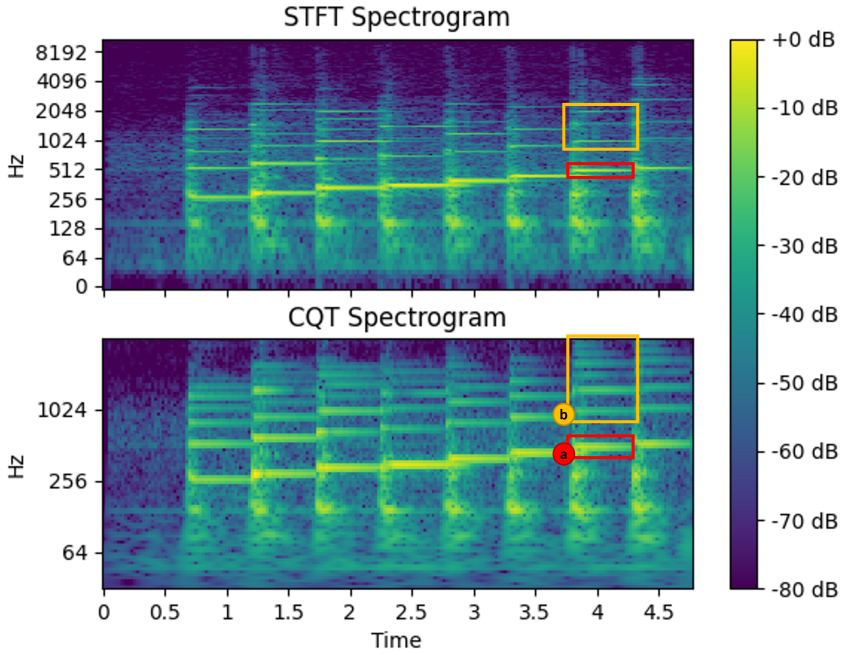

Figure 1: STFT and CQT spectrograms of a C-major piano scale. (a) CQT has even-thickness fundamental frequency bands. (b) STFT harmonics disappear at higher frequencies.

3.2 Model Compression

To design a system capable of real-time performance on a mobile device, I explore three different optimisation techniques for minimising model size and inference latency. In Section 5.1, I look at the impact of these techniques on inference time and accuracy.

3.2.1 Depthwise Separable Convolution

Depthwise separable convolutions are a method of reducing the size of individual convolutional layers in a CNN by splitting the convolution into smaller depthwise and separable convolutions.

Instead of defining a kernel that operates over all input channels to the layer (e.g., 3 x 3 x 3), a depthwise convolution layer instead separates this kernel into smaller separate kernels applied to each channel independently (e.g., 3 x 3 x 1), forming distinct intermediate layers. The outputs of these depth-wise convolutions are then combined with a separable convolution (i.e., a point-wise convolution) to introduce dependencies between the different layers (Figure 2).

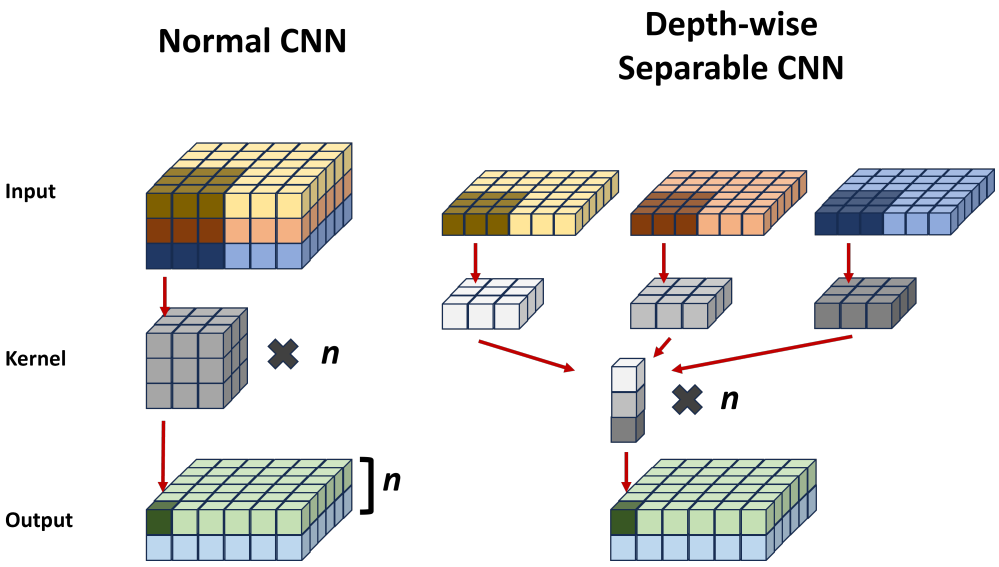

Figure 2: A comparison between CNNs and Depthwise Separable CNNs

By replacing convolutional layers with depthwise separable convolutions, we obtain a smaller and more efficient network, while maintaining information across the input channels [12].

3.2.2 Pruning

Pruning is a method of model compression by systematically removing certain parameters or even entire neurons. Pruning operates on the sparse network assumption that a large majority of network weights will have little impact on the model's predictions. Thus, by pruning these low-impact weights, we obtain a smaller network with performance comparable to the original.

For more effective pruning, it is more beneficial to prune after the model has been optimised, such that parameter weights can be taken as a measure of importance. As such, I choose to prune iteratively from an initial trained model, switching between tuning the network for some number of epochs and pruning more parameters.

3.2.3 Weight Quantisation

Quantisation is another method of model compression by changing the bit-width of the data structures used to store the model parameters. This reduction reduces the model's size, but can also make the model's inference quicker. Additionally, the most common form of quantisation, integer quantisation, which I implement in this paper, also allows the model to take advantage of existing hardware optimisations for integers.

Quantisation is most commonly done after training (i.e., training a model with float32 weights, then quantising to uint8), which I implement through TensorFlowLite [13].

4 Methods

4.1 Dataset

Considering the primary focus of this paper towards classical music identification, I decided to use a collection of classical pieces from RondoDB [14] as my training dataset. This dataset consists of 100 different classical pieces from a variety of composers and a multitude of instruments, with 5 recordings for each piece. For evaluation, the datasets used follow the literature, with the Mazurkas dataset [15] used to evaluate performance on classical pieces and the Covers80 [16] dataset used to evaluate the system's generalizability to popular music. The Mazurkas dataset consists of 49 different classical piano pieces (Mazurkas composed by Frédéric Chopin) with over 10 performers for each piece. The Covers80 dataset consists of 80 pop songs, containing one original recording and one cover for each. While the recordings for the Covers80 dataset are available outright, the recordings for the RondoDB and Mazurkas datasets were obtained by scraping through YouTube[2], taking the first matching result.

These recordings were averaged to a single channel and resampled to 22050 Hz, before being converted to CQTs using Librosa [17], using 12 bins per octave and a Hann window with a hop length of 512.

For training, the CQTs were further augmented to improve the model's capability to generalise (Algorithm 1). For a given batch of recordings, each recording was first converted into a CQT, and then compressed with a mean filter and down-sampled by 20 times. Then, tempo changes were simulated by stretching the CQT by a random changing factor r chosen uniformly between 0.7 and 1.3. Because most deep learning frameworks like TensorFlow require fixed-dimensional batched inputs, the entire batch of processed CQTs was then cropped to a length of either 200, 300, or 400 samples.

Algorithm 1 Training data augmentation

Require: $l \in \{200, 300, 400\}$
Ensure: B is a batch of recordings in dataset D
  for x in B **do**
    $r \leftarrow \mathcal{U}(0.7, 1.3)$
    $X \leftarrow \mathbf{CQT}(x)$
    $X \leftarrow$ stretch/squash by r
    $X \leftarrow$ crop random subsequence of length l
  end for

4.2 Models

The models produced in this paper were trained and compressed using TensorFlow [18]. Training and evaluation code can be found in the repository [19], along with the minimal files needed to reproduce the datasets used and a template Android app for deploying the models on mobile.

The initial CNN structure follows from Yu et. al. [11], which is shown in Figure 3. Training is done with the entire model, with the last dense layer being used for classification. For inference and evaluation, however, the output of the representation layer is instead extracted and used to compare the similarity between songs.

Additionally, I define a smaller model by replacing convolutional layers with depthwise separable convolutions. Both the normal and the smaller models were trained for 200 epochs on the RondoDB dataset, using the Adam optimiser with a learning rate of 0.001. This training took around 3 hours. For pruning, models were further trained for 5 epochs, iterating between training and pruning steps. Pruning was only applied to later convolutional layers, as the earlier layers are designed for extracting important melodic structures [11]. Quantisation was further applied after training and pruning.

In total, I trained and evaluated 8 different models, which are summarised below. Trained versions of these models (in .keras and .tflite formats) are available in the repository.

[2] using **youtube-dl**

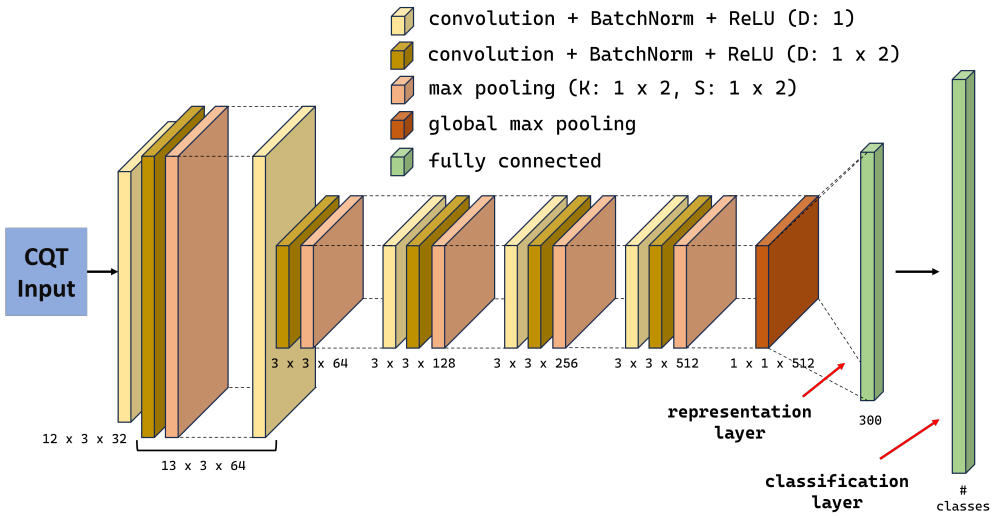

Figure 3: Architecture overview of the Normal Model

- Normal Model (CNN)
- Normal Model with Pruning
- Normal Model with Quantisation
- Normal Model with Pruning and Quantisation
- Smaller Model (Depth-wise Separable CNN)
- Smaller Model with Pruning
- Smaller Model with Quantisation
- Smaller Model with Pruning and Quantisation

5 Evaluation

I evaluate all models in terms of their classification accuracy, inference speed, and model size. As described in Section 4.1, I use the Mazurkas and Covers80 datasets to evaluate the performance of the models. For song identification performance, I adopt three commonly used metrics, the Mean Average Precision (MAP) (Equation 1), the Precision at 10 (P@10) (Equation 3), and the Mean Rank of the first correctly identified cover (MR).

Evaluation is done by first splitting the evaluation dataset into a reference and query dataset. For the Covers80 dataset, this is already done, as each of the 80 songs has a pre-defined original and cover. For the Mazurkas dataset, the reference recording is taken to be the performance by Arthur Rubinstein, as his performances are available for every Mazurka.

With a reference and query dataset, classification is done for a specific query song by first converting both the query and a reference to CQTs, running inference with the model on both, and calculating the similarity using the cosine distance between the two representation vectors. After doing this for all references in the dataset, references are ordered by increasing similarity, where the reference with the smallest cosine distance has the highest similarity and is the identified cover song. From this list of references with similarity scores, we can obtain our three evaluation metrics.

$$\text{MAP} = \sum_{n=1}^{n} \text{AP} \tag{1}$$

$$\text{AP} = \frac{\sum_{k=1}^{n} \text{Precision@}k \times I_{\text{k is relevant}}}{\text{total number of relevant songs}} \tag{2}$$

$$\text{P@10} = \frac{\text{total number of relevant songs in top 10}}{10} \tag{3}$$

Using these metrics, the evaluation results for the normal and smaller models on each dataset can be found in Table 4.

	Dataset	Evaluation Metrics		
		MAP	P@10	MR
Normal Model	Mazurkas	0.0665	0.00875	41.95
	Covers80	0.0561	0.00745	26.71
Smaller Model	Mazurkas	0.0590	0.005	34.74
	Covers80	0.0372	0.00750	45.11

Figure 4: Table of MAP, P@10, and MR for the normal and smaller models across datasets.

For each query song, we have that the dataset has one relevant song (the matching reference song), so we would expect a perfect model to have a MAP of 1, a P@10 of 0.1, and an MR of 1. However, we can see that both the normal and smaller model stray far from those optimal results, with MR scores around the middle of each dataset. Looking at the training loss for the normal and smaller models (Figure 5), the training behaviour explains the low performance of the models. We note that the training loss oscillates around 0.1, suggesting that the training dataset is not large enough for the models to learn anything.

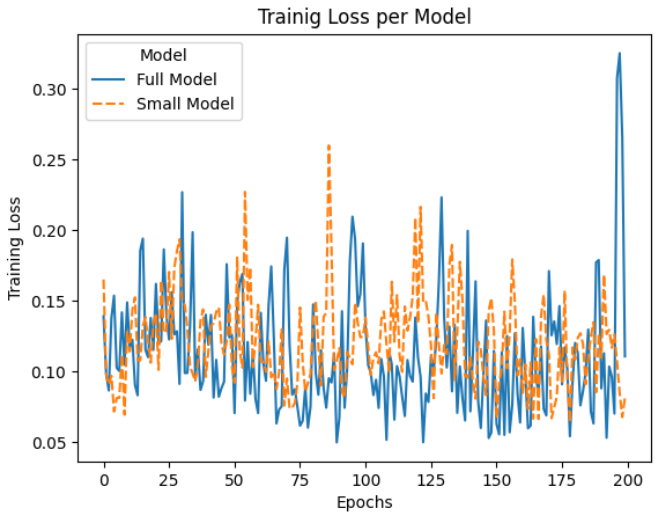

Figure 5: Training loss (sparse categorical cross-entropy) for the normal and smaller models.

This training dataset serves as the largest difference between prior research and this paper, as previous research has normally used the Second Hand Songs 100K dataset [10] for training, a much larger curated dataset of pop songs. However, considering the storage requirements and computational resources required to train using this dataset, the much smaller dataset in RondoDB was used instead for this project.

5.1 Model Compression

I then investigate the effects of pruning and quantisation on model size and inference speed. Model size is obtained using **tflite_tools**[3]. To measure the impacts of pruning on model size, the pruned models are also zipped, and the resulting file size is recorded [20]. Inference speed is calculated as the average of five runs on a fixed data sample. These results can be found in Tables 1 and 2.

We expect each of the compression methods to reduce the model efficacy, but improve the model's inference speed and memory efficiency. Looking at Table 1, this expected behaviour for inference speed and memory efficiency appears to be generally true. Comparing the techniques, we see the

[3]https://github.com/eliberis/tflite-tools

	Compression	Model Size		Inference Time (ms)
		Peak Mem. Usage (kB)	Zipped Size (MB)	
Normal Model	None	344	20.6	687
	Quantisation	**86** (-258)	4.4 (-16.2)	522 (-165)
	Pruning	344	20.6	723 (+36)
	Both	**86** (-258)	4.4 (-16.2)	520 (-167)
Smaller Model	None	344	13.2 (-7.4)	428 (-259)
	Quantisation	**86** (-258)	**2.81** (-17.79)	**311** (-376)
	Pruning	344	13.2 (-7.4)	453 (-234)
	Both	**86** (-258)	**2.81** (-17.79)	**311** (-376)

Table 1: Size and inference time for all models.

biggest drop in inference times when switching to the smaller model (i.e., depthwise separable convolutions), which is in line with prior research [12]. However, unexpectedly, pruning does not seem to give any benefits, both for memory usage and inference speeds. I hypothesise this is because the weights of the model are already sparse, meaning there will be very few non-significant weights to prune. This can explain both the poor performance of the model and the lack of compression found when applying pruning.

Given the poor performance of the base normal model, extrapolating the effects of these compression methods on identification performance is hard to justify; however, there is a general trend of performance decreases, which is consistent with what is expected of compression methods.

5.2 Real World Application

Even after compression techniques are applied, all models have sizes of several MB. Thus, they are too large for micro-controller applications. However, the relatively small sizes (as compared to larger neural networks) make them potentially usable for mobile applications. Furthermore, the use of a representation vector to calculate similarity means that the required reference database's size can be reduced to memory-efficient tensors instead of an entire audio signal. While the long inference times (relative to real-time applications) may hinder the model's usability, realistically, in a real-world music identification scenario, the model would be running inference once every several seconds, meaning the longer inference times are still reasonably usable.

6 Future Work

As mentioned in Section 5, I believe the poor model performance is primarily due to the smaller size of the training dataset as compared to other literature. Thus, future work would be to investigate the performance of the model given a larger or more complete dataset. Additionally, considering that classical music tends to have intrinsic structures in its composition (e.g., a sonata vs. a symphony are distinguishable), I would like to do further investigation on how generalisable different types of training datasets are to cover song identification. Similarly, the poor performance of the model opens an incentive to search for more data-efficient models of cover song identification capable of learning without requiring massive datasets or exploring the application of transfer learning from a pre-trained larger model.

Another important point of consideration is the compression techniques used in this paper. Molchanov et. al. [21] introduce the idea of pruning entire convolutional layers, rather than the conventional method of pruning weights. Given the lack of improvement from pruning in this paper and the large size of the original model, this direction could be considered for further compression.

	Compression	Evaluation Metrics		
		MAP	**P@10**	**MR**
Normal Model	None	0.0561	0.00745	**26.71**
	Quantisation	0.0419 (-.0142)	0.007 (-.00045)	33.99 (+7.28)
	Pruning	0.0474 (-.0087)	0.008 (+.00055)	32.66 (+5.95)
	Both	0.0810 (+.0249)	0.012 (+.00455)	31.42 (+4.71)
Smaller Model	None	0.0590 (+.0029)	0.005 (-.00245)	34.74 (+8.03)
	Quantisation	**0.104** (+.0479)	**0.017** (+.00955)	27.59 (+0.88)
	Pruning	0.0693 (+.0132)	0.008 (+.00055)	34.42 (+7.71)
	Both	0.0777 (+.0216)	0.014 (+.00655)	31.35 (+4.64)

(a) Results for the Mazurkas dataset.

	Compression	Evaluation Metrics		
		MAP	**P@10**	**MR**
Normal Model	None	**0.0665**	0.00875	41.95
	Quantisation	0.0621 (-.0044)	0.01125 (+.0025)	40.51 (-1.44)
	Pruning	0.0569 (-.0096)	0.00750 (-.00125)	42.11 (+0.16)
	Both	0.0575 (-.009)	0.01 (+.00125)	41.33 (-.62)
Smaller Model	None	0.0372 (-.0293)	0.00750 (-.00125)	45.11 (+3.16)
	Quantisation	0.0620 (-.0045)	**0.01250** (+.00375)	40.78 (-1.17)
	Pruning	0.0382 (-.0283)	0.00625 (-.0025)	45.75 (+3.8)
	Both	0.0632 (-.0033)	0.01125 (+.0025)	**40.09** (-1.86)

(b) Results for the Covers80 dataset.

Table 2: Evaluation metrics, model size, and inference speeds for the normal and smaller model under multiple compression techniques.

Finally, I would like to complete more realistic testing of the system (i.e., in the presence of noise or multiple songs playing simultaneously). Unfortunately, time constraints on the project meant I was unable to fully implement an Android app capable of running inference using these models but creating such an app would allow me to test such effects in the context of live inference and further explore usability and latency concerns with the models.

7 Conclusion

In this work, I perform a thorough analysis of the effects of multiple model compression techniques on the ability of CNNs to identify cover songs. My results provide evidence to support the application of depth-wise separable convolutions as a compression technique for CNNs, as well as the feasibility of compressed neural networks for cover song identification in mobile applications.

References

[1] Avery Wang. An industrial strength audio search algorithm. In *International Society for Music Information Retrieval Conference*, 2003. URL: https://api.semanticscholar.org/CorpusID:5515593.

[2] Joren Six and Marc Leman. Panako - a scalable acoustic fingerprinting system handling time-scale and pitch modification. In *International Society for Music Information Retrieval Conference*, 2014. URL: https://api.semanticscholar.org/CorpusID:17146219.

[3] Shumeet Baluja and Michele Covell. Waveprint: Efficient wavelet-based audio fingerprinting. *Pattern Recognition*, 41(11):3467–3480, 2008. URL: https://www.sciencedirect.com/science/article/pii/S0031320308001702, doi:https://doi.org/10.1016/j.patcog.2008.05.006.

[4] Mathieu Ramona and Geoffroy Peeters. Audioprint: An efficient audio fingerprint system based on a novel cost-less synchronization scheme. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 818–822, 2013. doi:10.1109/ICASSP.2013.6637762.

[5] Reinhard Sonnleitner and Gerhard Widmer. Robust quad-based audio fingerprinting. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(3):409–421, 2016. doi:10.1109/TASLP.2015.2509248.

[6] Andreas Arzt and Gerhard Widmer. Piece identification in classical piano music without reference scores, 2017. arXiv:1708.00733.

[7] Stephan Rein and Martin Reisslein. Identifying the classical music composition of an unknown performance with wavelet dispersion vector and neural nets. *Information Sciences*, 176(12):1629–1655, 2006. URL: https://www.sciencedirect.com/science/article/pii/S0020025505001878, doi:https://doi.org/10.1016/j.ins.2005.06.002.

[8] Rahul Rama Varior, Mrinal Haloi, and Gang Wang. Gated siamese convolutional neural network architecture for human re-identification, 2016. arXiv:1607.08378.

[9] Aapo Hakala. *Song Identification From Live Music Using Siamese Convolutional Neural Networks*. 2023. URL: https://trepo.tuni.fi/handle/10024/149299.

[10] Xiaoshuo Xu, Xiaoou Chen, and Deshun Yang. Key-invariant convolutional neural network toward efficient cover song identification. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018. doi:10.1109/ICME.2018.8486531.

[11] Zhesong Yu, Xiaoshuo Xu, Xiaoou Chen, and Deshun Yang. Learning a representation for cover song identification using convolutional neural network. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 541–545, 2020. doi:10.1109/ICASSP40776.2020.9053839.

[12] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. arXiv:1610.02357.

[13] Giorgos Demosthenous and Vassilis Vassiliades. Continual learning on the edge with tensorflow lite, 2021. arXiv:2105.01946.

[14] RondoDB. Rondodb. URL: https://www.rondodb.com/about.

[15] CHARM. The mazurka project. URL: http://www.mazurka.org.uk/.

[16] Daniel Ellis and Courtenay Cotton. The 2007 labrosa cover song detection system. 01 2007.

[17] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in Science Conference*, pages 18 – 24, 2015. doi:10.25080/Majora-7b98e3ed-003.

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete

Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[19] Vivian Li. Project repository, 2024. URL: https://github.com/VivianDLi/music-classification.

[20] Pruning with keras. URL: https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras.

[21] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning, 2019. arXiv:1906.10771.