A FULLY AUTOMATED MUSIC EQUALIZER BASED ON MUSIC GENRE DETECTION USING DEEP LEARNING AND NEURAL NETWORK

Kevin Hu1, Yu Sun2, Yujia Zhang3

1Sage Hill School, 20402 Newport Coast Dr, Newport Beach, CA 92657
2California State Polytechnic University, Pomona, CA, 91768,
Irvine, CA 92620
3University of California Irvine, Irvine, CA 92697

ABSTRACT

Recent years have witnessed the dramatic popularity of online music streaming and the use of headphones like AirPods, which millions of people use daily [1]. Melodic EQ was inspired by these users to create the best audio listening experience for listeners with various preferences [2]. Melodic EQ is a project that creates custom EQs to the user's custom music tastes and filters the audio to fit their favorite settings. To achieve this goal, the process starts with a song file taken from an existing file, for example, Spotify downloads or mp3s. This file is then uploaded to the app. The software sorts the song in a genre detecting Algorithm and assigns a genre label to that song. Inside the app, the user will create or select EQs for that genre and apply it to their music. The interface is easy to use and the app aims to make everyone's preferences achievable and on the fly. That's why there are presets for each category for users who are unfamiliar with equalizers, and custom settings for advanced users to create their perfect sound for each genre.

KEYWORDS

AI auto genre detection, Automatic genre switching, EQ, Convolution music equalizer network

1. INTRODUCTION

Audio has been an integral part of human society for millennia [3]. Ranging from conversations to music, it is integral to humans as group animals to communicate and express their feelings through music [4]. Now more than ever, people have access to all types of audio material. Recently there has been a meteoric rise of personal audio devices, headphone use, and entertainment viewing, phones, headphones, and other audio broadcasting devices are in every corner of society. This trend is also increasingly encouraged by corporations through YouTube, streaming services, and music streaming [5]. With so many different individuals around the world, everyone has their favorite listening experience in their corner of the audible frequency spectrum. EQs are the perfect solution to that problem, Melodic EQ steps this solution up by adding modern technology in neural networks and personalized user inputs to give every individual their own personalized best audio listening experience. I made Melodic EQ under the

mentorship of Jonathan Thamrun, Product Support Associate at Nodus Technologies, and Yu Sun, Associate Professor of Computer Science at Cal Poly Pomona.

David C. Wyld et al. (Eds): CCSEA, AIFU, EMSA, NLCAI, NCOM, SIPRO, SEA, DKMP, BDML, BIoT, CLOUD - 2023
pp. 201-209, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130517

Some techniques and systems that allow users to control EQs intuitively include built-in hardware settings for headphones, background software, and manual adjustment of EQ inside individual apps. However, these proposals do not possess genre recognition or switching to individual EQs for each setting [6]. Their implementations are also limited in scale, for example, limiting to only pairing with specific apps, scenarios, or static settings. An example of hardware settings is the Sony app. They are limited to only a static EQ pairing with the headphones the user has, and cannot be transferred to another pair. The method is simply a standard EQ and doesn't have any AI inputs. Then we move on to more advanced software like Nahmic 3, the method/algorithm used can already identify between scenarios like video calls and listening to music, however, the results often cannot satisfy the need for switching EQs fitting multiple scenarios in a short period like a YouTube video or movie scene that has music and dialogue crossed over constantly. Melodic EQ is more focused on identifying genre and then switching up the EQ. Some EQs for universal apps are also very specific to target quality speakers and headphones which might lead to buzzing. With Melodic EQ, we allow users to customize settings for each device they play their music on, which makes life a lot more simple as to not overdrive cheap quality speakers that cannot fit into the one-size fits all product.

In this paper, I followed the same line of research by Automatic Music Genre Classification Based on CRNN [7]. My goal is to integrate an AI algorithm working with equalizer to sort music, and I was inspired by their research to use certain features of their research. First, they used the GTZAN dataset with a CRNN in Python to make a neural network to predict the music genre. I was able to find the same exact dataset on kaggle to also use this dataset to create my own algorithm. Secondly, they made the data readable as an image using spectrograms, or images of the soundwaves. I too needed to use spectrograms and had to use the PyDub library with the Librosa library in python to edit the audio, get channel lengths, and spectrograms. Lastly, they used a CRNN as their main priority but we used a Feedforward Neural Network or FNN, which they compared to. They were able to get a similar accuracy using a STFN or Short-time Fourier transform, a similar feedforward network.

To test the working of the app along with the backend, we used a combination of techniques that increases the accuracy of the model. First, to prove the results of my neural network, I visualized the train-test split, where 80 percent of the GTZAN dataset was used to train the feedforward neural network and 20 percent of the dataset was used to test the accuracy of the algorithm. I visualized the dataset by using pandas to graph the accuracy of the neural network. Next I also used the keras built in model.summary() to check the accuracy of the algorithm which gives a summary of all the components and the accuracy of both the training and the testing of the dataset. Finally, in the actual app, I was able to import different genres of songs from hip-hop, country, and pop to test the working of the app working with the algorithm.

The rest of the paper is organized as follows: section 2 gives the details on the challenges that we met during the experiment and how those challenges influenced the designing of the app. Section 3 focuses on the details of the methodology of the app, including the backend and frontends corresponding to the challenges and goals that we mentioned in the previous sections. Section 4 presents the relevant details about the experiment we did, the various methods of testing and evaluation, and graphs and figures of the data we collected. Section 5 shows related works that were an inspiration and parallel studies based of the project. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. License to Access Music

My biggest challenge with this app was the ability to access music because we do not have the license to access music through streaming services who have dedicated servers to store millions of audio files [8]. This is the biggest limitation to the app because it makes it so much harder to work with all forms of audio being processed by the user's phone. Unfortunately, this means the app cannot select any song off of an online platform and just plug Melodic EQ's algorithm onto it. The user can only play audio from a file picker for downloaded audio files which severely limited the usability of the app. However, if a user paid for a streaming service and downloaded mp3s, the user could go to the folder of the app and stream music directly from their downloads and we could only rely on this method being an independent 3rd party option. There is no queue for the app in beta either so it is just a gimmick without better implementation for more audio modifications and playing.

2.2. Using A New Coding Language in Android Studio

My second biggest challenge was using a new coding language in Android Studio by using Flutter [9]. Dart is the official Flutter language and I have never used it before in any scenario. Dart is very similar to Java but I had little to no experience in it so I had to start everything from the ground up. Fortunately, my mentor Jonothan was very adept in it and was able to help me code. The hardest integral part of the coding was inheritance, which involved multiple classes with object oriented code spanning the entire project. All of the variables, and instances got very messy very quickly and I had to spend hours fixing the code. What made this process even more challenging was the audio processing using Flutter, which involved multiple independent packages made by other programs to develop the logic and the backend. These are not able to be explained by the normal methods of coding in Dart, and rely on documentation and custom objects and functions. I had to follow my mentor Jonathan very closely so as to not get lost, and successfully played audio with equalizer filters when coding the app.

2.3. Never-Ending Grind to Perfect the Algorithm

My last challenge was the never-ending grind to perfect the algorithm and sort the correct genre. The music genre detecting algorithm is the most vital part of the audio sorting process, so the genre detection needs to acquire the highest test accuracy possible. To make this happen I had to adjust and learn multiple ways to sort the preprocessing functions, and the parameters of the neural network to get the best results. This meant expanding the libraries by many folds and ultimately finding the best ones through repeated testing. However, there was also the problem of overfitting or underfitting data [10]. Underfitting means that the model makes accurate, but initially incorrect predictions where both train error and val/test error is large. Overfitting means that the model makes false predictions because train error is very small but val/test error is large. After adjusting the preprocessing, dropout rate, and epoch number I was able to get an optimal fitting number.

3. SOLUTION

Melodic EQ is a custom equalizer that uses an AI algorithm and user input to filter all types of audio. The process starts with an independent algorithm that sorts the music into genres without any inputs, the genre detected is then sent to the algorithm. The algorithm is a feedforward neural network which is a fully connected deep learning network with multiple units. A unit in layer n receives input from all units in layer n-1, and sends output to all units in layer n+1, so there are no loops in the hidden layers (fig 2). Finally, the app checks if there is a pre-existing custom equalizer for that genre and applies a custom or preset equalizer onto the music when it is played (fig 1).

For the neural network, the experiment used Python to create and train a neural network on Google Collab. For the training of the neural network, the dataset was taken from Kaggle to find open-source prelabeled data for music genre classification. The "gtzan-dataset-music-genre-classification," was chosen because of its labeled data from 10 different genres and fit the criteria for most genres of music. The experiment used this dataset with a feedforward neural network by using the python library Pydub, Librosa, and tensorflow [15]. Using the PyDub with the Librosa library python can get certain information about the file. Librosa helps with this method by extracting the log cepstrum or Mel-Frequency Cepstrum Coefficients (MFCC) as input. Log cepstrum is the logarithm operation after the Fourier transform of the signal, and then perform the inverse Fourier transform to obtain the spectrogram. The feedforward neural network was fed the MFCC data along with hyperparameters to get the best results. Feedforward Neural Networks are fully connected, and use dense layers, which are a classic fully connected neural network layer where each input node is connected to each output node. These layers also have a dropout attribute attached to them so that when the layer is used, the activations are set to zero for some random nodes and prevent overfitting (fig 6). This tensorflow model was then run for 200 epochs with a batch size of 32 and was able to have a test accuracy of 94.5% and a valid accuracy of 66% (fig 3). Very similar to the research paper we were inspired by and also outperformed the CRNN in that paper. This wraps up the backend calculations and moves along to the front end. For the frontend, the app was developed on Android Studio using Flutter with Dart as the programming language. I created a GUI for the user to interact with the app which needed to choose the song to equalize, adjust the presets, and adjust settings of the actual app, which makes 3 different pages: Home, Equalizer, and Settings (fig 4). Everything depends on the filters in the Equalizer tab, the equalizer tab has 10 presets for the different types of genres the algorithm sorts into and is a one-size fits all solution for non advanced users who just want a better listening experience. Advanced users are also not left out and can use this page to create their own presets for each of the 10 genres if they find something that suits them more. These equalizer settings work with the home page where the users input their music through a file picking process of the downloaded music on their device. We integrated the neural network into a web service scheme with a Python Flask framework which listens to the web request from the app frontend and sends the music file back to the users. We use an AWS server to host the whole service to make the api call stable.

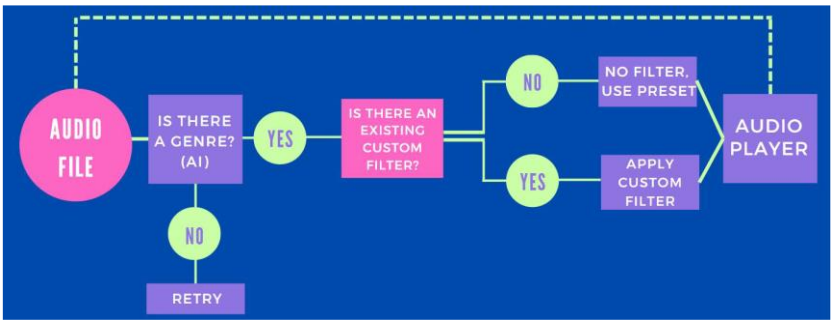

Figure 1. Overview of the solution

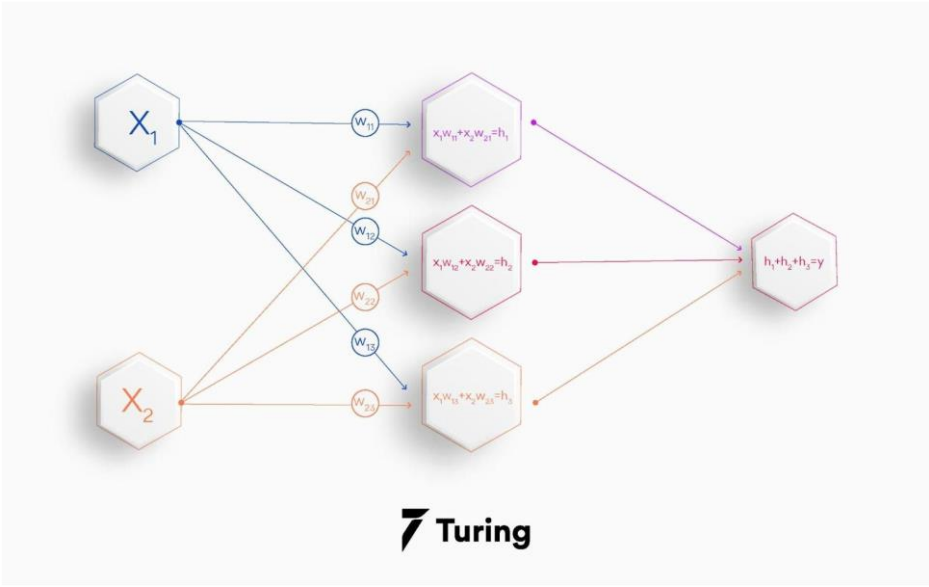

Figure 2. Hidden layers

```
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 200
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath=f'saved_models/audio_classification_{current_time}.h5f5',
                               verbose=1, save_best_only=True)

start = datetime.now()

history = model.fit(X_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test, y_test), callbacks=[checkpointer], verbose=1)

duration = datetime.now() - start
print('Training complete in time: ', duration)

25/25 [==============================] - 0s 9ms/step - loss: 0.2196 - accuracy: 0.9387 - val_loss: 2.8662 - val_accuracy: 0.6700
Epoch 187/200
22/25 [=======================>....] - ETA: 0s - loss: 0.1871 - accuracy: 0.9503
Epoch 187: val_loss did not improve from 1.34838
25/25 [==============================] - 0s 9ms/step - loss: 0.1839 - accuracy: 0.9499 - val_loss: 2.8161 - val_accuracy: 0.6350
Epoch 188/200
22/25 [=======================>....] - ETA: 0s - loss: 0.1695 - accuracy: 0.9474
Epoch 188: val_loss did not improve from 1.34838
25/25 [==============================] - 0s 9ms/step - loss: 0.1820 - accuracy: 0.9424 - val_loss: 3.1807 - val_accuracy: 0.6700
Epoch 189/200
24/25 [===========================>..] - ETA: 0s - loss: 0.1869 - accuracy: 0.9453
Epoch 189: val_loss did not improve from 1.34838
25/25 [==============================] - 0s 9ms/step - loss: 0.1848 - accuracy: 0.9462 - val_loss: 3.1572 - val_accuracy: 0.6550
```

Figure 3. Screenshot of code

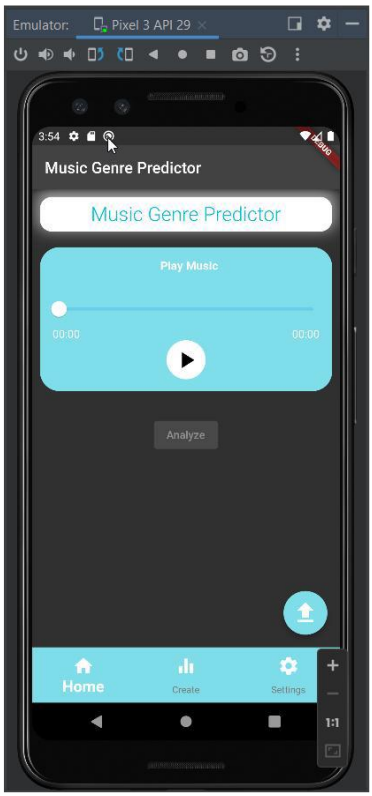

Figure 4. Screenshot of main page

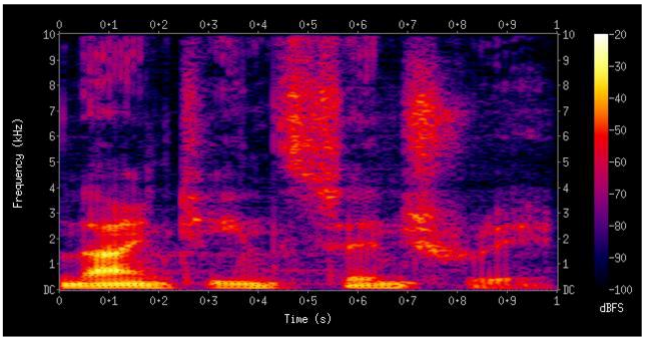

Figure 5. Frequency vs time

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1024)              41984

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 512)               524800

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 128)               32896

 dropout_3 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_4 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 32)                2080

 dropout_5 (Dropout)         (None, 32)                0

 dense_6 (Dense)             (None, 10)                330

=================================================================
Total params: 741,674
Trainable params: 741,674
Non-trainable params: 0
_________________________________________________________________
```

Figure 6. Sequential model screenshot

4. EXPERIMENT

4.1. Experiment 1

The first experiment is built into tensorflow, where the values of the loss, accuracy, val_loss, and val_accuracy are given in a dataset, which can then be plotted as a graph. The sample size of the train test split is 80 percent training, and 20 percent testing, so with this ratio we can see any overfitting or underfitting of the dataset or overly high or low training accuracies that do not translate to better val_accuracy [14].

This is a graph of the experiment we performed (fig 7). we can see that there is overfitting of the graph after about 50 epochs as the loss went down but the val_loss increased. This means we needed to make adjustments in the epochs as it started overfitting after a 50 epochs.

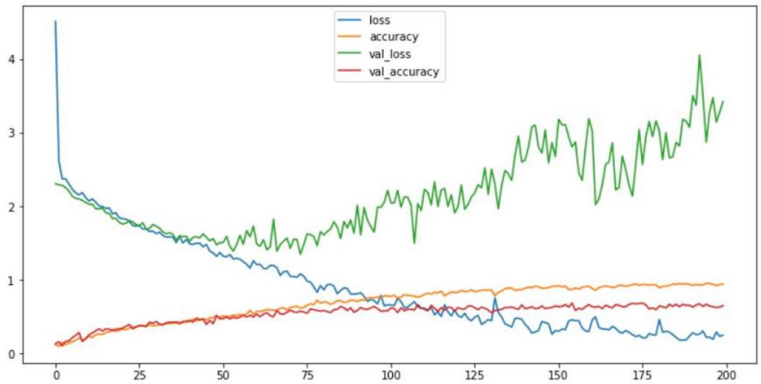

Figure 5. Voice vs mass 4.2. Experiment 2

The next experiment is inside of the android studio test, where we input two songs from two different genres like “Blank Space” by Taylor Swift, and m.A.A.d city by Kendrick Lamar. They test the algorithm and the app by sending the song to the server and run the algorithm, as well as testing the app’s ability to play and equalize the audio.

By seeing the output we can see that it was sorted and the process of uploading the audio, using the neural network, and getting the genre classification back works. This also shows that the app was able to receive the genre, and apply a filter. And most importantly we can hear that an equalizer was able to be applied.

The experiment addresses the problems above by figuring out if the feedforward neural network was actually having good accuracy for sorting the genre. Ultimately, it was able to prove some overfitting of the data, but also show little error for obvious music choices.

5. RELATED WORK

The 3D-DCDAE: Unsupervised Music Latent Representations Learning Method Based on a Deep 3D Convolutional Denoising Autoencoder for Music Genre Classification by Lvyang Qiu, Shuyu Li and Yunsick Sung was used to inspire the use of the GTZAN dataset on kaggle as well as combining some of the seven features they used such as MFCC, spectral roll-off, zero-crossing rate, chroma frequency, and rhythm histogram [11].

Automatic Music Genre Classification Based on CRNN by Yu-Huei Cheng, Member, IAENG, Pang-Ching Chang, Duc-Man Nguyen, and Che-Nan Kuo helped the experiment out by testing out the limits of CRNNs and Short-time Fourier transform to compare the accuracies [12]. By basing off their research, we were able to choose feedforward neural networks as more accurate model for classifying the music.

Comparing the Accuracy of Deep Neural Networks (DNN) and Convolutional Neural Network (CNN) in Music Genre Recognition (MGR): Experiments on Kurdish Music Aza Zuhair and Hossein Hassani helped us with the feature extraction and possible future implementations of data collection [13]. The feature extraction introduced the librosa library to extract things such as MFCC, as well as the shape of the spectral envelope into duration and segments for the neural network to include.

6. CONCLUSIONS

In the future, we hope that we can input streaming and cross app streaming into Melodic EQ so that more apps and audios can be processed in the background during all uses of audio. This can be achieved by asking for permission to constantly run in the background. The most important step would be detecting if any audio is playing and then taking the audio from the other app and then incorporating it into the Melodic EQ app and sending it to the server and back. Lag would need to be cut down and other problems such as compatibility with audio would also need to be addressed. However, if this works in the future, the app would be a non-interfering user experience that would only enrich the user’s experience with audio.

Current limitations include the accuracy of the dataset, as even more advanced neural networks are not achieving higher accuracy than simple feed forward loops. Until then there are limitations in software to get better accuracy. Another important step is the integration of streaming into the app, which will likely never happen. This severely limits the amount of songs that can be played

and the usability of the app in a simple manner. Lastly, is the user's trust if we implement background running, many users are protective of their privacy and we need to build that trust to allow the app to work in the background.

In the future, I hope to increase the accuracy as software improves and better optimization and parameters help with the accuracy. Finding better labeled music data might also help as music evolves as time moves on, ultimately the end goal is to improve the accuracy of the model with no lag.

REFERENCES

[1] Datta, Hannes, George Knox, and Bart J. Bronnenberg. "Changing their tune: How consumers'adoption of online streaming affects music consumption and discovery." Marketing Science 37.1 (2018): 5-21.

[2] VÂRLAN, Petre Marcel. "THE EMOTIONAL QUOTIENT – A FACTOR OF MUSICAL KNOWLEDGE." Bulletin of the Transilvania University of Brasov, Series VIII: Art & Sport 5 (2012).

[3] Zhu, Hao, et al. "Deep audio-visual learning: A survey." International Journal of Automation and Computing 18.3 (2021): 351-376.

[4] Roy, William G., and Timothy J. Dowd. "What is sociological about music?." Annual Review of Sociology 36 (2010): 183-203.

[5] Prey, Robert. "Nothing personal: algorithmic individuation on music streaming platforms." Media, Culture & Society 40.7 (2018): 1086-1100.

[6] Bar-On, Reuven. "Emotional and social intelligence: Insights from the Emotional Quotient Inventory." (2000).

[7] Silla, Carlos N., Alessandro L. Koerich, and Celso AA Kaestner. "A machine learning approach to automatic music genre classification." Journal of the Brazilian Computer Society 14.3 (2008): 7-18.

[8] Wlömert, Nils, and Dominik Papies. "On-demand streaming services and music industry revenues — Insights from Spotify's market entry." International Journal of Research in Marketing 33.2 (2016): 314-327.

[9] Esmaeel, Hana R. "Apply android studio (SDK) tools." International Journal of Advanced Research in Computer Science and Software Engineering 5.5 (2015).

[10] Van der Aalst, Wil MP, et al. "Process mining: a two-step approach to balance between underfitting and overfitting." Software & Systems Modeling 9.1 (2010): 87-111.

[11] Qiu, Lvyang, Shuyu Li, and Yunsick Sung. "3D-DCDAE: Unsupervised music latent representations learning method based on a deep 3d convolutional denoising autoencoder for music genre classification." Mathematics 9.18 (2021): 2274.

[12] Cheng, Yu-Huei, et al. "Automatic Music Genre Classification Based on CRNN." Engineering Letters 29.1 (2020).

[13] Zuhair, Aza, and Hossein Hassani. "Comparing the accuracy of deep neural networks (DNN) and convolutional neural network (CNN) in music genre recognition (MGR): experiments on Kurdish music." arXiv preprint arXiv:2111.11063 (2021).

[14] Koehrsen, Will. "Overfitting vs. underfitting: A complete example." Towards Data Science (2018).

[15] Abadi, Martín. "TensorFlow: learning functions at scale." Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming. 2016.