GenreWiz: Music Genre Classification with RNN and CNN

Live Demo | Github Repo

Due to free Heroku account, please allow a short delay before the webpage starts up

Team
Li Yuxuan 1003607
Keith Ng 1003515
Ng Jia Yi 1003696

Introduction

The rise in need for automatic music genre classification (AMGC) revolves around the fact that artists today tend to distribute their songs on their websites, making music database management a must. Another recent development is to consume music via streaming, raising the tendency for online radio stations to play songs based on genre preferences. In addition, music browsing and playlist creation these days all utilizes recommendations based on genre.

In this project, we seek to classify music into its respective genre by passing a music spectrogram into a parallel convolutional neural network (CNN) - recurrent neural network (RNN) model.

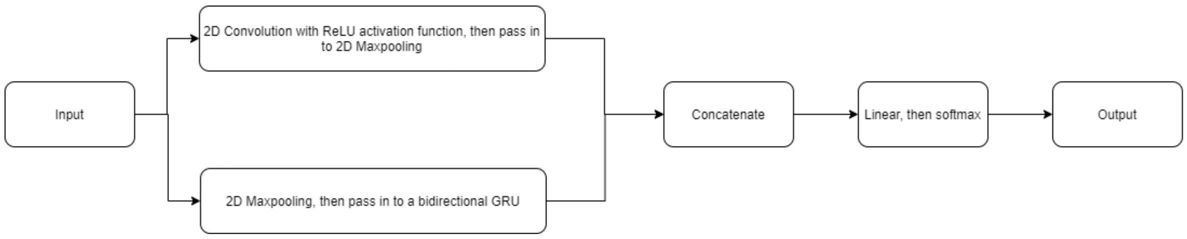
Model

In our original project proposal, we suggested the usage of a CNN model as we intended to convert audio files into spectrograms - a visual representation of audio across frequency and time dimension. CNN makes perfect sense here since our data inputs into the model are essentially images, and CNN excels at image classification.

On the other hand, RNN excels in understanding sequential data by making the hidden state at time t dependent on hidden state at time t-1. The spectrograms have a time component and RNNs can do a much better job of identifying the short term and longer term temporal features in the song.

We referenced the following post from Medium (https://towardsdatascience.com/using-cnns-and-rnns-for-music-genre-recognition-2435fb2ed6af) (Priya, 2018).

We implemented the parallel CNN-RNN model as specified in the above post:


Input
2D Convolution with ReLU activation function, then pass in to 2D Maxpooling
2D Maxpooling, then pass in to a bidirectional GRU
Concatenate
Linear, then softmax
Output

```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input (InputLayer)              (None, 640, 128, 1)  0
________________________________________________________________________________________________
conv_1 (Conv2D)                 (None, 638, 128, 16) 64          input[0][0]
________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 319, 64, 16)  0           conv_1[0][0]
________________________________________________________________________________________________
conv_2 (Conv2D)                 (None, 317, 64, 32)  1568        max_pooling2d_1[0][0]
________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 158, 32, 32)  0           conv_2[0][0]
________________________________________________________________________________________________
conv_3 (Conv2D)                 (None, 156, 32, 64)  6208        max_pooling2d_2[0][0]
________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 78, 16, 64)   0           conv_3[0][0]
________________________________________________________________________________________________
conv_4 (Conv2D)                 (None, 76, 16, 64)   12352       max_pooling2d_3[0][0]
________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 19, 4, 64)    0           conv_4[0][0]
________________________________________________________________________________________________
conv_5 (Conv2D)                 (None, 17, 4, 64)    12352       max_pooling2d_4[0][0]
________________________________________________________________________________________________
pool_lstm (MaxPooling2D)        (None, 160, 64, 1)   0           input[0][0]
________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 4, 1, 64)     0           conv_5[0][0]
________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 160, 64)      0           pool_lstm[0][0]
________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 256)          0           max_pooling2d_5[0][0]
________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 128)          49536       lambda_1[0][0]
________________________________________________________________________________________________
concat (Concatenate)            (None, 384)          0           flatten_1[0][0]
                                                                 bidirectional_1[0][0]
________________________________________________________________________________________________
preds (Dense)                   (None, 8)            3080        concat[0][0]
================================================================================================
```

Parallel CNN-RNN Model

The convolutional block of the model consists of 2D convolution layer followed by a 2D Max pooling layer. The block is repeated 5 times. The final output is flattened.

The recurrent block starts with 2D max pooling layer of pool size (4,2) to reduce the size of the spectrogram before LSTM operation. This feature reduction was done primarily to speed up processing. The reduced image is sent to a bidirectional GRU with 64 units. .

The outputs from the convolutional and recurrent blocks are then concatenated . Finally we have a dense layer with SoftMax activation.

Here is our PyTorch model :

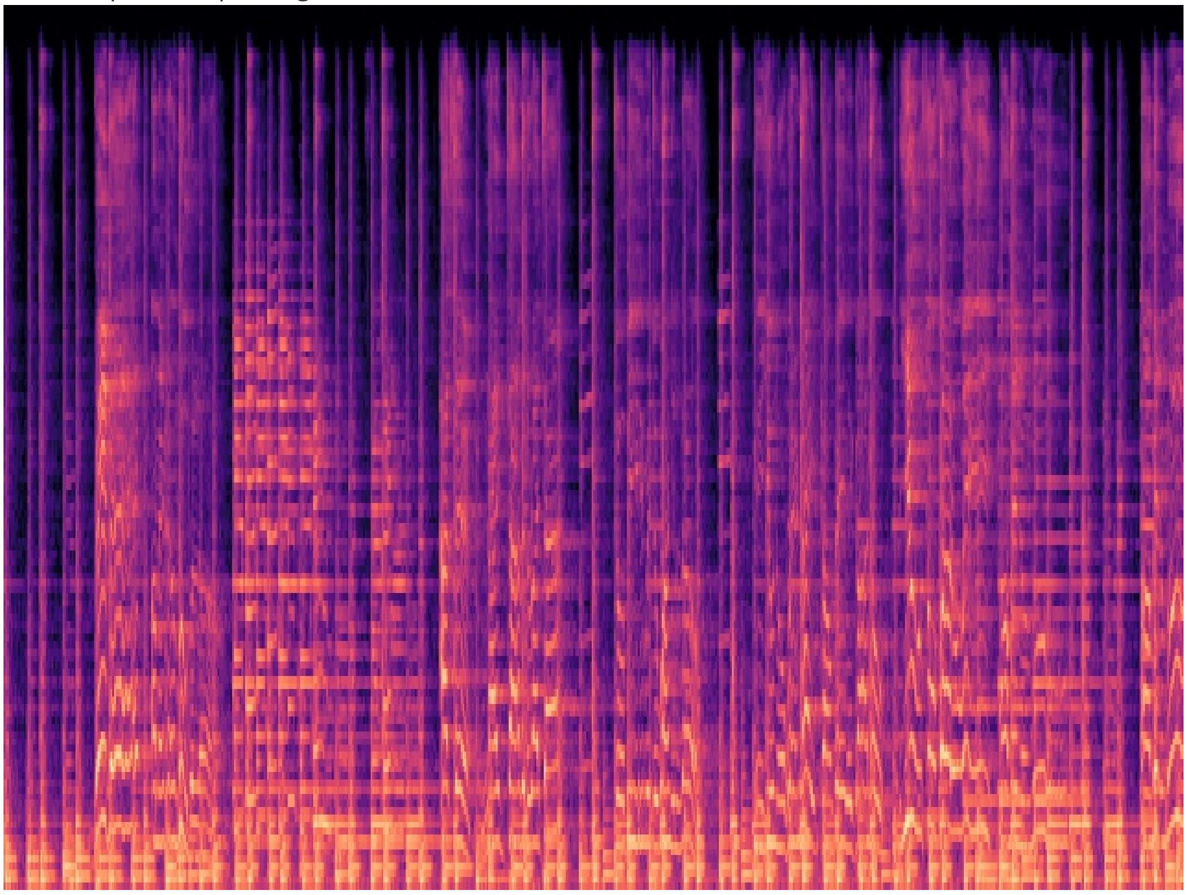
```
class PCRNN(nn.Module):
    def __init__(self, ks=(3, 1)):
        super().__init__()
        # CNN
        self.conv2d_1 = nn.Conv2d(1, 16, ks)
        self.conv2d_2 = nn.Conv2d(16, 32, ks)
        self.conv2d_3 = nn.Conv2d(32, 64, ks)
        self.conv2d_4 = nn.Conv2d(64, 64, ks)
        self.conv2d_5 = nn.Conv2d(64, 64, ks)

        self.maxPool2d_1 = nn.MaxPool2d((2, 2))
        self.maxPool2d_2 = nn.MaxPool2d((2, 2))
        self.maxPool2d_3 = nn.MaxPool2d((2, 2))
        self.maxPool2d_4 = nn.MaxPool2d((4, 4))
        self.maxPool2d_5 = nn.MaxPool2d((4, 4))

        self.flatten_1 = nn.Flatten()

        # RNN
        self.pool_lstm = nn.MaxPool2d((4, 2))
        self.gru = nn.GRU(input_size=64, hidden_size=1, bidirectional=True)

        # Overall
        self.linear = nn.Linear(576, 10)  # 10 genres
        self.softmax = nn.Softmax(dim=1)

    def forward_cnn(self, x: torch.Tensor):
        x = F.relu(self.conv2d_1(x))
        x = self.maxPool2d_1(x)
        x = F.relu(self.conv2d_2(x))
        x = self.maxPool2d_2(x)
        x = F.relu(self.conv2d_3(x))
        x = self.maxPool2d_3(x)
        x = F.relu(self.conv2d_4(x))
        x = self.maxPool2d_4(x)
        x = F.relu(self.conv2d_5(x))
        x = self.maxPool2d_5(x)
        x = self.flatten_1(x)
        return x

    def forward_rnn(self, x: torch.Tensor):
        x = self.pool_lstm(x)
        x = x.view(x.size(0), x.size(2), x.size(3))
        x = self.gru(x)
        x = self.flatten_1(x[0])
        return x

    def forward(self, x: torch.Tensor):
        cnn = self.forward_cnn(x)
        rnn = self.forward_rnn(x)
        output = torch.cat([cnn, rnn], dim=1)
        output = self.linear(output)
        output = self.softmax(output)

        return output
```

Implementation

Data loader

We created our customized DataLoader class that is able to read the WAV audio files from `data/` folder and transform them into the Mel-spectrograms in the form of a 2D array for input to the model. Taking into account that audio file length might vary slightly, even though each song sample is about 30 seconds in total, we did some simple cropping of the sample to only take the first 640 units of the spectrogram. This can standardize our input shape to the model as well. The dataloader follows the 8-2 proportion when dividing the dataset into training and validation. Hence, 80% of the samples are used for training and 20% of the samples are used for validation.

An example of a spectrogram looks as such:



We designed the code such that the DataLoader is a simple plug-and-play function that can be used at any stage.

In later development, we also included `test` mode in the same DataLoader. Hence, when switched to `test` mode, this DataLoader will grab the test samples from another directory and output the corresponding 2D spectrogram data as well as the corresponding label.

Training and Validation

We used the Kaggle GTZAN dataset as our training and validation data. The dataset consists of 1000 songs split evenly between 10 music genres, The .wav audio clip(30s snippet) and spectrograms were provided for each song. However, we did not utilize their provided spectrogram as we were unable to replicate it with our own songs.

We split the dataset into 80% training data and 20% validation data. Meaning to say, within each genre, 80 songs were used for training, and 20 songs for validation. The selections were done randomly.

The model was trained using RMSProp optimizer with a learning rate of 0.0005 and the loss function was categorical cross entropy. The model was trained for 120+ epochs and Learning Rate was constantly changed (increased when accuracy plateaued and decreased when accuracy fluctuated too much).

During training, the model that refreshes the highest accuracy will be automatically saved. Hence we make sure that we always obtain the best model after the training.

Testing

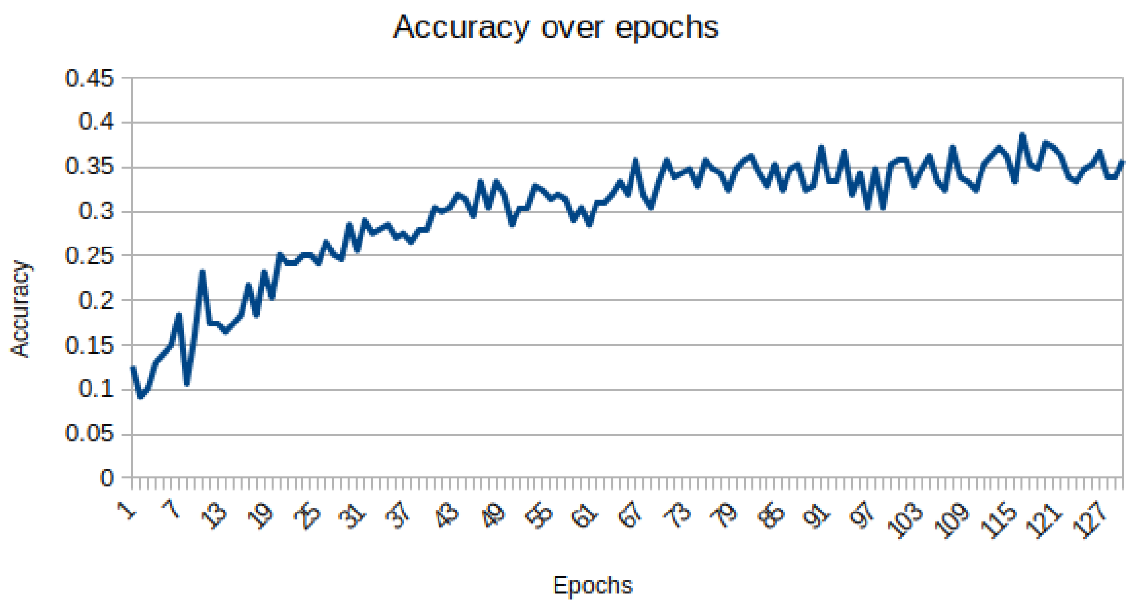
For the testing data, we went to YouTube and downloaded 5 songs from each genre of music. In the GTZAN dataset, the songs ranged from the 1970s to the early 2000s. It consisted a mix of recorded songs and live performance. There was also no consistency in the volumes of the songs or the sections of the songs - some were choruses, some were verses, some were instrumentals and some were a mix. Thus, when selecting when preparing the songs for the test data, we ensured that they were as diverse as the ones provided in the GTZAN dataset.

We have an evaluation script which runs the trained model through the 50 test samples we collected and generated a classification report as well as a confusion matrix. The test results are discussed in the next section.

Results

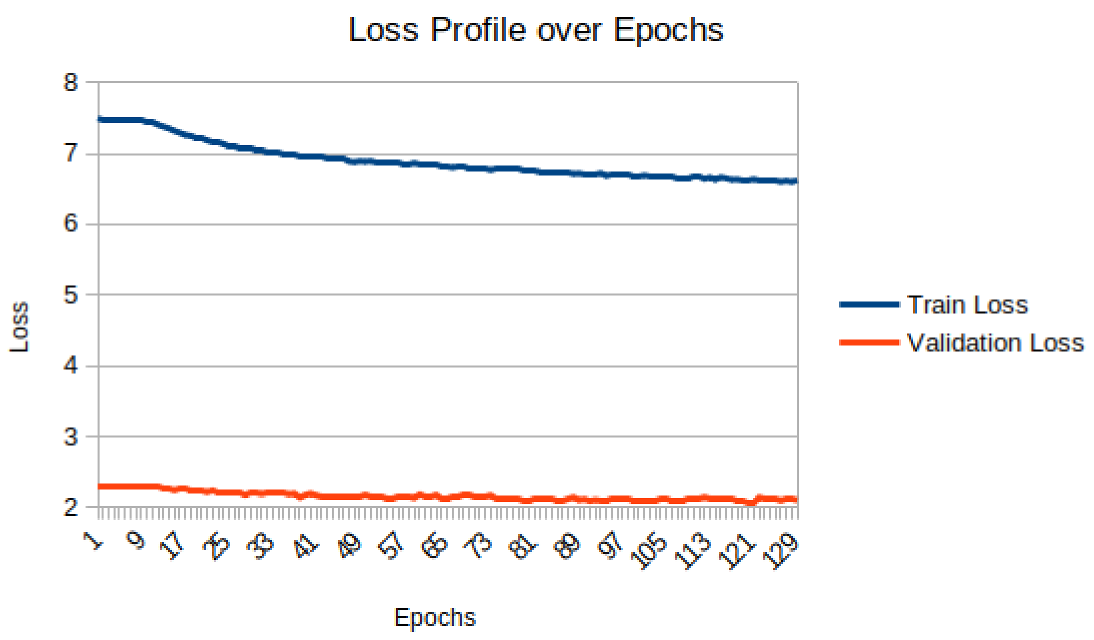
Accuracy profile

Here is the accuracy profile on validation:



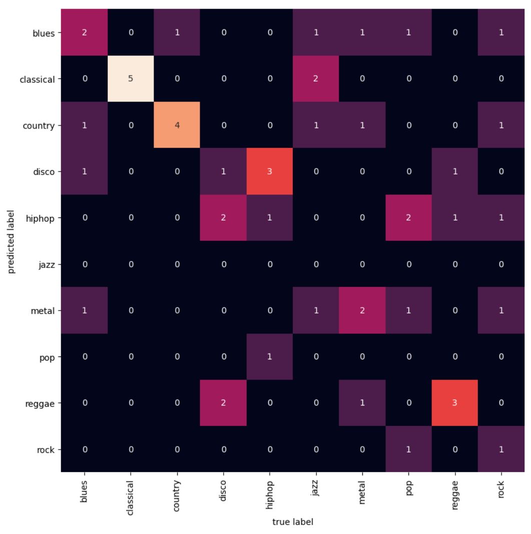
Loss profile

Here is the loss profile on train and validation:



Confusion matrix

We ran our model on 50 test samples that we downloaded from online. Here is the confusion matrix of the model performance. We can see that Classical, Country and Reggae were mostly correctly classified whereas Disco, HipPop, Jazz, Pop and Rock were hard to differentiate.



Classification Report

Here is the classification report generated on the 50 test samples.

```
             precision    recall  f1-score   support

      blues       0.29      0.40      0.33         5
  classical       0.71      1.00      0.83         5
    country       0.50      0.80      0.62         5
      disco       0.17      0.20      0.18         5
     hiphop       0.14      0.20      0.17         5
```

```
        jazz       0.00      0.00      0.00         5
       metal       0.33      0.40      0.36         5
         pop       0.00      0.00      0.00         5
      reggae       0.50      0.60      0.55         5
        rock       0.50      0.20      0.29         5

    accuracy                           0.38        50
   macro avg       0.31      0.38      0.33        50
weighted avg       0.31      0.38      0.33        50

F1: 0.332534
Recall: 0.380000
Precision: 0.314286
Accuracy: 0.380000
```

Results analysis

From the aforementioned results generated, we can see that our model's performance is not fantastic. This can be due to several factors.

Firstly, our training dataset is limited. We only have 1000 songs as input and 800 are used for training and 200 are used for validation. This means that our model is only trained based on the 800 songs, which is not enough.

Secondly, our test samples are found online by ourselves on YouTube, which contain our personal judgement of whether a song belongs to which genre. Hence, our test samples might not be all correctly labelled in the first place. A better strategy if time and resources permit, is to find professional music curator to help to label the test samples to their correct genres. In addition to that, there are no strict boundaries that specify a song genres. Songs could be a mixture of different genres and hence, they might not have one true label.

Therefore, with more training data and more accurately labelled data (multi labelled classification), and perhaps more epochs of training, we foresee an improvement in our model performance.

Graphical User Interface

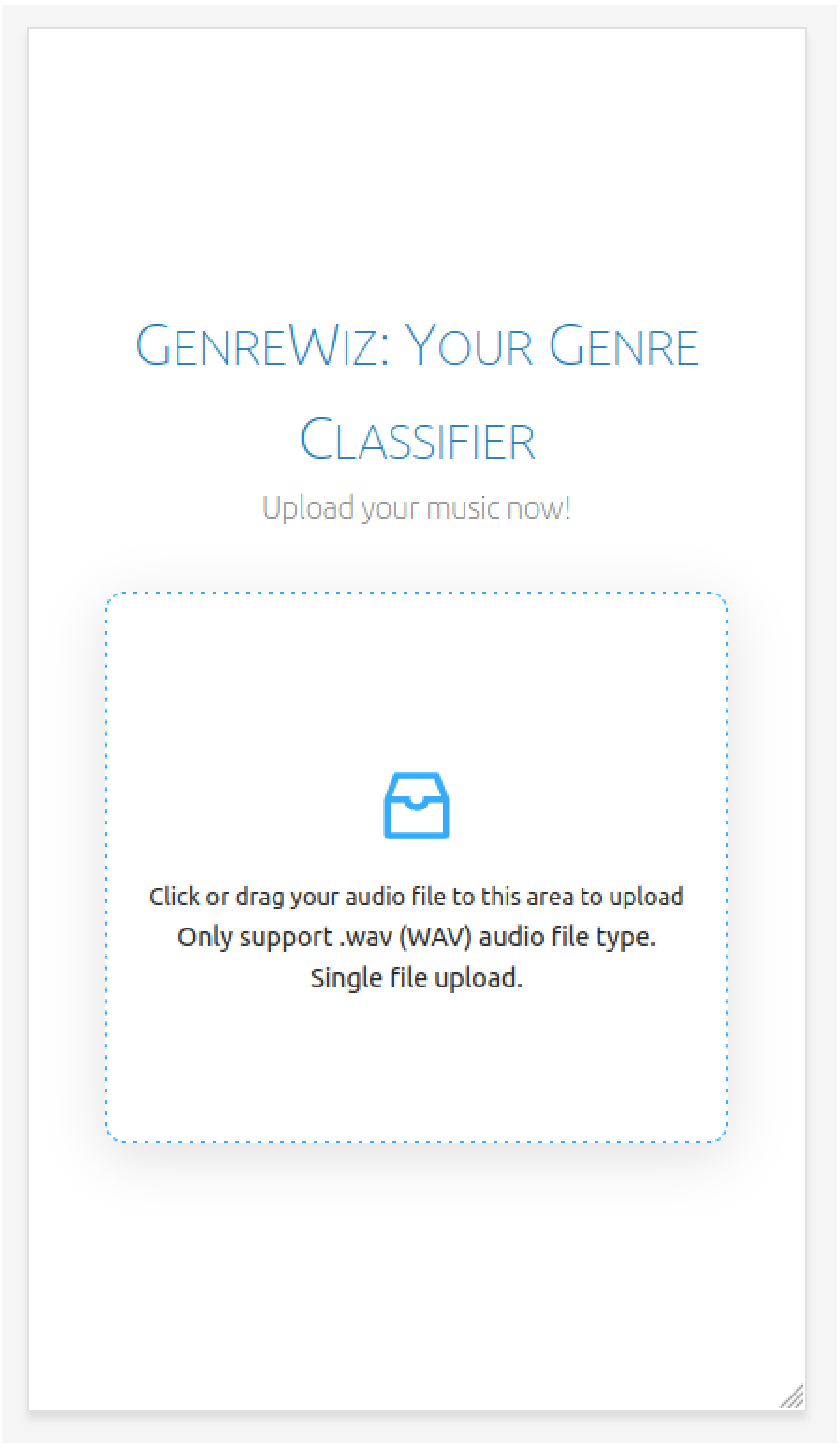
We have developed a GUI demo for our Genre Classifier in React, and hosted publicly on Heroku. Due the the free account limitation, after 30 minutes of inactivity, Heroku will shutdown the server. Hence, users may experience a long wait when accessing the website.

Here is the public link to our web demo: https://genrewiz.herokuapp.com

In this web demo, we provide the users a simple feature of uploading their WAV audio file and the backend model will predict and output the top 5 possible genres that it identifies.

We have also provided some wav audio samples in the file /sample for you to try out.

Here are the snapshots of our web demo, both on PC and on mobile devices:


GenreWiz: Your Genre Classifier

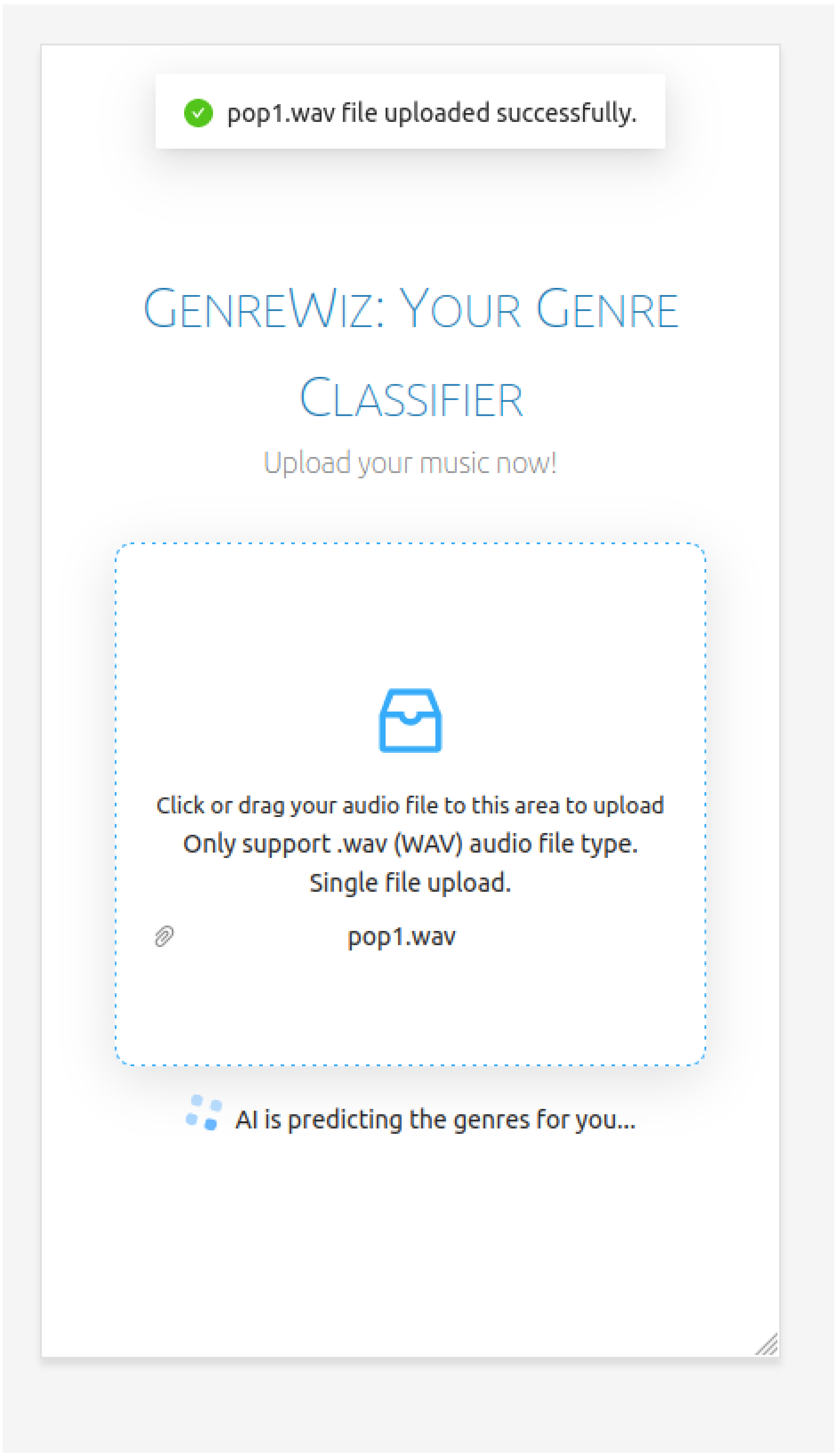
Upload your music now!

Click or drag your audio file to this area to upload

Only support .wav (WAV) audio file type.

Single file upload.

Figure 1: Mobile homepage of GenreWiz


pop1.wav file uploaded successfully.

GenreWiz: Your Genre Classifier

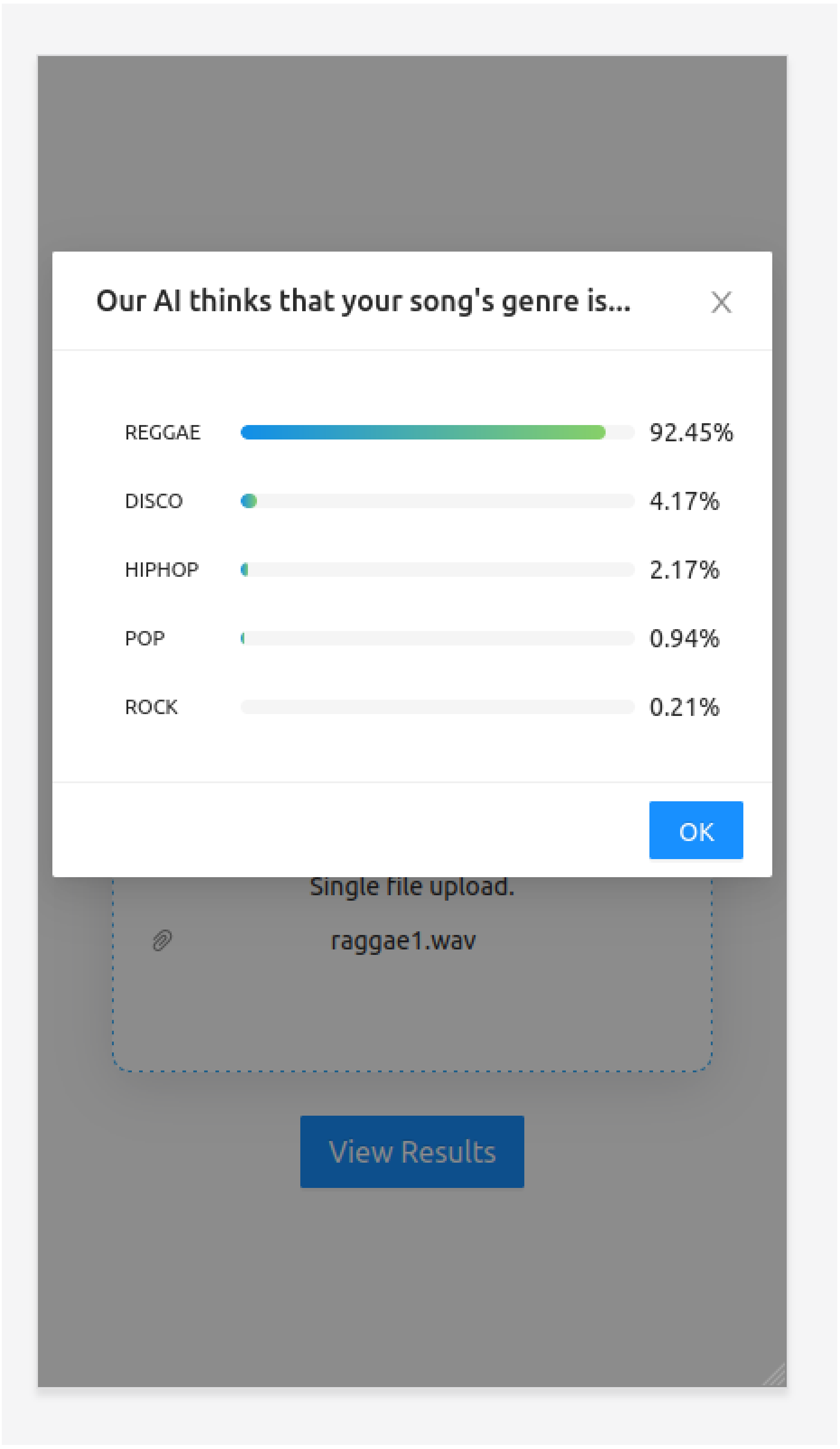
Upload your music now!

Click or drag your audio file to this area to upload
Only support .wav (WAV) audio file type.
Single file upload.

pop1.wav

AI is predicting the genres for you...

Figure 2: Upload WAV file successfully and start prediction


Our AI thinks that your song's genre is...

REGGAE	92.45%
DISCO	4.17%
HIPHOP	2.17%
POP	0.94%
ROCK	0.21%

Figure 3: View prediction results

Conclusion

For our project, we have used a model that uses both CNN and RNN in parallel. We take in .wav music audio files as input, and convert them into visual spectrograms before passing it to our model. Though our accuracy and f1 score we have obtained for our test set is not as high as we had hoped to be, when we compare our model with the state-of-the-art accuracy by the medium post (51%), we feel that it was indeed a good learning process, and the results we have obtained might be limited due to the aforementioned reasons.