Prog-rock vs. the world classifier

Jaemoon Lee
Kyle Lo

LSTM model

1. Techniques

Information extractions

To make an input for LSTM training, we convert a song file into timeseries multidimensional array with shape $(length, embedded\ size)$ where $length$ is the length of the song and $embedded\ size$ is size of embedded array for each element corresponding to a timestamp of a song. To model the embedded array of each element, we use Librosa (https://librosa.github.io/librosa/) library to extract features of a song, including the following features:

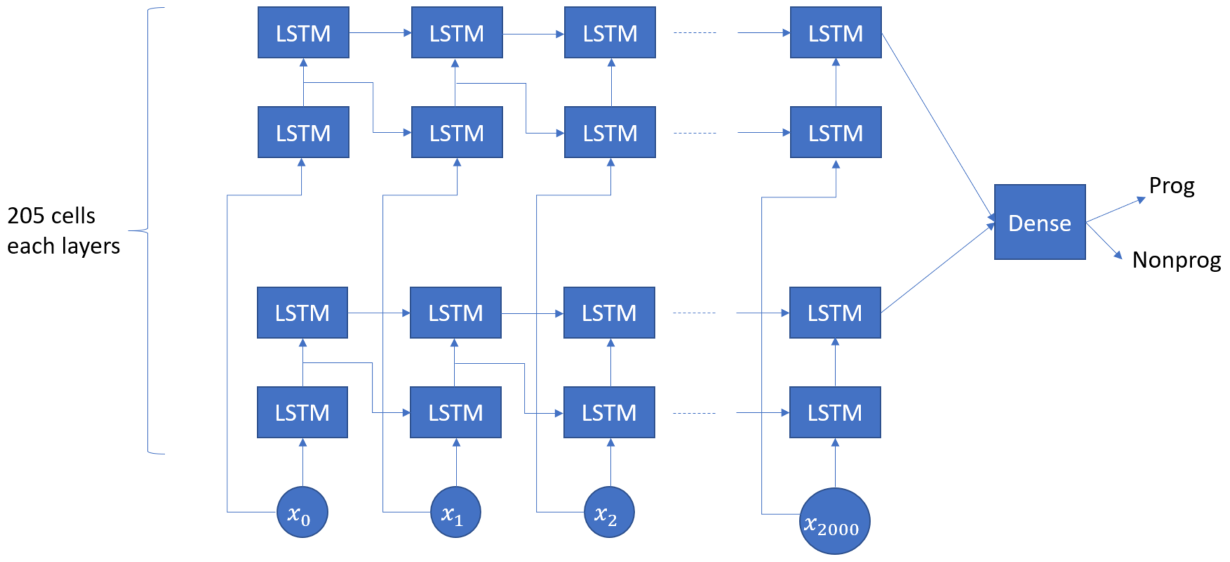
- Mel-frequency ceptral coefficients (MFCCs)
- Chroma_stft
- Chroma_cqt
- Chroma_cens
- Melspectrogram
- Rms
- Rmse
- Spectral centroid
- Spectral_bandwidth
- Spectral_contrast
- Spectral_flatness
- Spectral_rolloff
- Poly_features
- Tonnetz
- Zero_crossing_Rate

Refer https://librosa.github.io/librosa/ for more detail. Therefore, each timeseries element will be modelled by a 205-dimensional array.

To vary the data for training and validate, we segment a song file into small segments such that each segment has length at most 2000. We do "sequential" segmentation by extracting segments start from indices of multiplication of 1000 (i.e. 0, 1000, 2000 etc.), which guarantees the segments overlaps. In overall, the song is represented by $(no\ segments, 2000{,}205)$ arrays where $no\ segments$ is the number of segments of the song.

Training model

We use a 2-layer LSTM model, each layer contains 205 LSTM cells. The architecture is as follows:



With the first layer, we set $return\ sequence = True$, which guarantees that the LSTM cells returns all of the outputs from the LSTM cells through time. Thus, the output of the first layer is $(2000{,}205)$. In contrast, we set this parameter is $False$ in second layer, which then only return the output of the last element in timeseries. Thus, the output of the second layer is (205), which then is put into $Dense$ convolution layer for classifying Prog and Nonprog.

To justify the model, the reason we use 2 layers is to allow for greater model complexity, the second LSTM layer can help to identify high-level feature representation of the input data. Because of second layer appearance, the first layer should return sequences. To be honest, due to lack of computational capacity, we cannot add more layers to the model (more layer means more training parameters, causing slower training).

To avoid overfitting, we use 2 hyperparameters for each layer. To be specific, we set the $dropout$ to be 0.05 and $current\ dropout$ to be 0.35. There is no explanation for this setting except that we refer from available sources and (again) due to lack of computational capacity, we could not test with different settings of these hyperparameters.

Evaluation

To predict a song whether it is Prog or Nonprog, we extract the information from the song into $(no\ segments, 2000{,}205)$ – which is the same as we do with training set. Then the model will make prediction on each segment of the songs. The final result is decided in 2 ways

- Majority vote: The genre of the song is the one with more than 50% segments is predicted to be.

- Mean score: Each segment is associated with s_p – fitness score of the segment to be Prog and s_n – fitness score of the segment to be Nonprog $(s_p + s_n = 1)$. We then take average of s_p and s_n for all segments and return the genre which got higher average score.

2. Results

We let the training run "forever" (no. epochs is 1,000,000) and save the model after each epoch. The following results is the best one we have achieved for each set.

With the validation set (21 prog songs and 71 nonprog songs), here is the confusion matrix of the results

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	15/21	6/21
Actual: Nonprog	13/71	58/71

Table 1: Validation set with majority vote

Table 1 summarizes prediction of validation set using majority vote. In overall, the accuracy on prog song prediction is 71.43%, nonprog is 81.69% and overall 79.35%.

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	15/21	6/21
Actual: Nonprog	11/71	60/71

Table 2: Validation set with mean score

Table 2 summarizes prediction of validation set using mean score. In overall, the accuracy on prog song prediction is 71.43%, nonprog is 84.51% and overall 81.52%.

With the training set, here is the confusion matrix of the results

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	73/73	0/73
Actual: Nonprog	23/302	279/302

Table 3: Training set with majority vote

Table 3 summarizes prediction of training set using majority vote. In overall, the accuracy on prog song prediction is 100%, nonprog is 92.38% and overall 93.87%.

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	73/73	0/73
Actual: Nonprog	20/302	282/302

Table 4: Training set with mean score

Table 4 summarizes prediction of training set using mean score. In overall, the accuracy on prog song prediction is 100%, nonprog is 93.38% and overall 94.67%.

With the test set, here is the confusion matrix of the results

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	87/102	15/102
Actual: Nonprog	41/102	61/102

Table 5: Test set with majority vote

Table 5 summarizes prediction of test set using majority vote. In overall, the accuracy on prog song prediction is 85.29%, nonprog is 59.80% and overall 73.27%.

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	91/102	11/102
Actual: Nonprog	43/102	59/102

Table 6: Test set with mean score

Table 6 summarizes prediction of test set using mean score. In overall, the accuracy on prog song prediction is 89.22%, nonprog is 57.84% and overall 74.26%.

Explanation

To explain why our model makes error in prediction, we look into returns score of each segments of a song. For example, the song titled "01. Folklore.mp3", here are fitness scores of this song's segments (format is $[s_n\ s_p]$)

```
[[6.4910877e-01 3.5089117e-01]
 [9.7818267e-01 2.1817369e-02]
 [8.5402572e-01 1.4597426e-01]
 [7.2991955e-01 2.7008042e-01]
 [8.1055695e-01 1.8944302e-01]
 [8.7174428e-01 1.2825571e-01]
 [6.7841852e-01 3.2158145e-01]
 [2.9690057e-01 7.0309943e-01]
 [6.5223789e-01 3.4776211e-01]
 [4.7382507e-01 5.2617490e-01]
 [9.9996138e-01 3.8585778e-05]
 [9.9795377e-01 2.0461932e-03]
 [2.5185245e-01 7.4814755e-01]
 [3.1274110e-01 6.8725890e-01]
 [4.1204900e-01 5.8795094e-01]
 [4.9358773e-01 5.0641221e-01]
 [5.2018911e-01 4.7981086e-01]
 [5.8234292e-01 4.1765702e-01]
 [4.4586360e-01 5.5413640e-01]]
```

Our model made "wrong" prediction on the first 1/3 of the song. However, when we listen to the song, those 1/3 has slow rhythm, mainly electric organs (we guess?) – which could be confused with other music genres. In contrast, our model makes good prediction on the final 1/3 of the song which has fast tempo and mainly formed by electric guitars sound. This phenomenon also happens in the song "03. Egoist Hedonist.mp3" where we think major part of the songs has slow and even rhythm, which may impact our model's decision. A solution to improve the model may be to split a song into parts which really feature Progressive rock attribute and make prediction on it. Regarding to Nonprog misclassification, a typical mistake of our model is to classify songs with fast tempo and electric guitars into prog song (e.g. the song “06--Pulp-Fiction-Misirlou.mp3” in validation set). We have to admit that our lack of music knowledge is a major hindrance for us to achieve better model.

2. CNN model

1. Why try CNN in this project

Expect for using LSTM to classify the prog and non-prog music, we also give CNN (convolution neuron network) a try. The reason is that music can also be represented as 2D image and CNN is well-known for its powerful ability in image classification. Nowadays CNN has become one of the most critical research in machine learning field. Many inspirational applications such as imageNet [1] and Alpha Go [2] are all build on CNN. Therefore, by implementing the CNN-based music classifier, we hope that we can get familiar with the principles behind the scene.

2. The Design of CNN Model

Number and size of layers

Figure 1 demonstrates the structure of CNN model we design in this project. This setting is inspired by the genre classification paper by Y. Han et. al. [3]. The authors extract Mel-spectrogram as input patterns and use these patterns to train the CNN model as a seven genres classifier.

At the beginning, we implemented exact the same model with paper [3]. However, the model achieves accuracy over 90% on training set but less than 60% on validation set. This means the model tends to be over fitting. The variables in the original model are too many for the case of two-class classification. The unnecessary variables provide too much ability for model to fit the training patterns, which causes the model lack of generalization.

Therefore, we abandon two convolution layers with 256 filters and reduce the size of fully connected layers after flatten operation from 1024 to 32. We try the fully connected layer with 16, 32, 64 units and 32 units provides the best performance. We think this is because 64 units make the model easily to be overfitting; in contrast, 16 units is too few for model to memorize how to separate important features passing from convolution layers.

Dropout

To prevent model from overfitting, we also try to use dropout in our model. As shown in the following table, the accuracy increases by 7% after adding the dropout. Dropout randomly shutdown the neurons to stop updating its weighting in each iteration. This results that neurons updates theirs weights using different patterns. And this enforces the neurons in the same layer to learning different characteristics.

Model	Prog		Non prog		Overall ACC	
	All Song	Windows	All Song	Windows	All Songs	Windows
Proposed CNN	74/101	37049/59900	75/101	20358/30681	73.76%	63.38%
Proposed CNN w/o Dropout	40/101	27379/59900	89/101	23469/30681	63.86%	56.13%

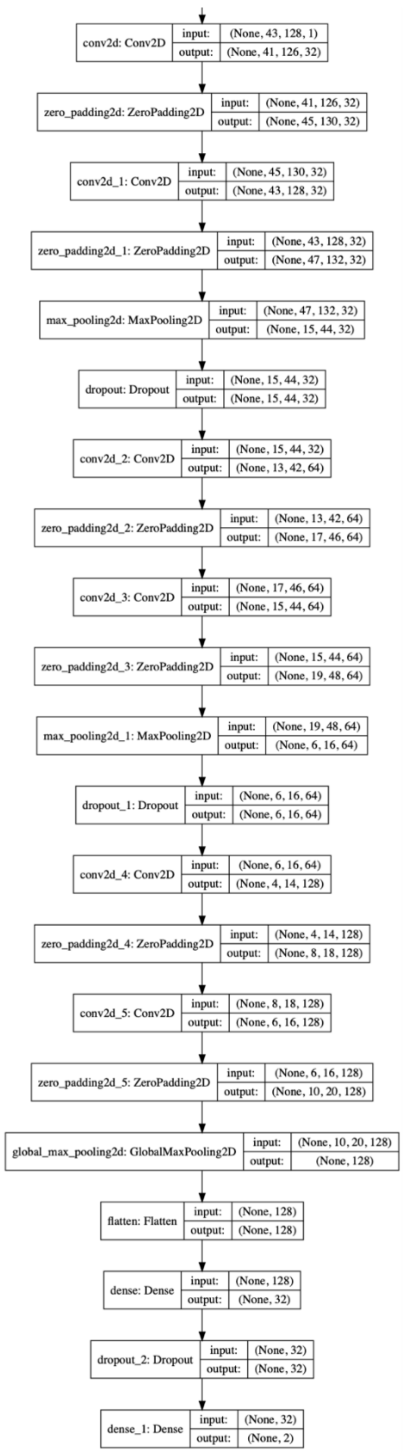

conv2d: Conv2D
input: (None, 43, 128, 1)
output: (None, 41, 126, 32)
zero_padding2d: ZeroPadding2D
input: (None, 41, 126, 32)
output: (None, 45, 130, 32)
conv2d_1: Conv2D
input: (None, 45, 130, 32)
output: (None, 43, 128, 32)
zero_padding2d_1: ZeroPadding2D
input: (None, 43, 128, 32)
output: (None, 47, 132, 32)
max_pooling2d: MaxPooling2D
input: (None, 47, 132, 32)
output: (None, 15, 44, 32)
dropout: Dropout
input: (None, 15, 44, 32)
output: (None, 15, 44, 32)
conv2d_2: Conv2D
input: (None, 15, 44, 32)
output: (None, 13, 42, 64)
zero_padding2d_2: ZeroPadding2D
input: (None, 13, 42, 64)
output: (None, 17, 46, 64)
conv2d_3: Conv2D
input: (None, 17, 46, 64)
output: (None, 15, 44, 64)
zero_padding2d_3: ZeroPadding2D
input: (None, 15, 44, 64)
output: (None, 19, 48, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 19, 48, 64)
output: (None, 6, 16, 64)
dropout_1: Dropout
input: (None, 6, 16, 64)
output: (None, 6, 16, 64)
conv2d_4: Conv2D
input: (None, 6, 16, 64)
output: (None, 4, 14, 128)
zero_padding2d_4: ZeroPadding2D
input: (None, 4, 14, 128)
output: (None, 8, 18, 128)
conv2d_5: Conv2D
input: (None, 8, 18, 128)
output: (None, 6, 16, 128)
zero_padding2d_5: ZeroPadding2D
input: (None, 6, 16, 128)
output: (None, 10, 20, 128)
global_max_pooling2d: GlobalMaxPooling2D
input: (None, 10, 20, 128)
output: (None, 128)
flatten: Flatten
input: (None, 128)
output: (None, 128)
dense: Dense
input: (None, 128)
output: (None, 32)
dropout_2: Dropout
input: (None, 32)
output: (None, 32)
dense_1: Dense
input: (None, 32)
output: (None, 2)

Fig. 1. The proposed CNN model

Learning rate

There is an interested found when we are training the original CNN model in paper [3]. We think this is the "vanishing gradient problem" mentioned in the lecture of ResNet. We try to set larger learning rate (0.01) to train the model. Everything works fine for the simpler CNN such as model with 32x2 conv layers, 32x2 + 64x2 conv layers, and 32x2 + 64x2 + 128x2 conv layers. However, when we implement the deeper model with 32x2 + 64x2 + 128x2 + 256x2 conv layers, the loss remains unchanged after third epochs.

We think the unchanged gradient is caused by gradient explosion. This happens when the network is so deep (12 conv layers) and the learning rate is too high. We solve this problem by reducing learning rate to 0.001.

3. Training of CNN model

Mel-spectrogram

In the proposed CNN model, Mel-spectrogram is used as the format of our input patterns. The shape of each input pattern is 43x128, where 43 represents the numbers of samples in one second (the window size is one second) and 128 is the number of bins in frequency domain.

The training set are generated from the songs given in the class, which has 75 prog and 305 non-prog songs. Since the beginning of the song are usually quiet and not representative, we also cutoff the first 30 seconds for each song. When we are shifting the window, we do not let the window overlap with the previous one. This is because the window size is only one second, enough training set be generated without overlap.

The total training patterns for prog songs is 40927 and for non-prog songs is 68763, which is 109690 in total.

TPU vs GPU vs CPU

Since training more than 100 thousand data is really slow on the laptop, we also seek for the more powerful computational resource on the net. Colab [4] by Google is an amazing platform that provides us free TPU and K80 GPU to use. On our Laptop (i7 - 6 cores) takes more than 10 minutes to finish one epoch. If we use 14 folded-cross validation, the computational time will be more than 2 days (14 folds x 25 epochs x 10 mins = 58 hours) Then, if we try our model on GPU, each epoch only takes about 3 mins. Using GPU can cut down the training time to less than one day. TPU is more powerful which only takes 20 seconds per epochs (20X faster). With the help of TPU, we can try more different model in a short time.

4. Experimental results

Results for training

Model	Learning rate	Epochs	Batch size	Training ACC
Proposed CNN	0.001	25	1024	85.35%

Results for testing

Model	Prog		Non prog		Overall ACC	
Proposed CNN	All Song	Windows	All Song	Windows	All Songs	Windows
	74/101	37049/59900	75/101	20358/30681	73.76	63.38

Misclassified prog as non-prog songs

- 02 - Xanadu.mp3
- 08 Suite Sister Mary.mp3
- Simple Boy.mp3
- 07. ShutDOWN.mp3
- 05_Turn of the Century.mp3.
- Devin Townsend Project - Kingdom.mp3
- 04_Birthright.mp3
- The Power To Believe-King Crimson-07-The Power To Believe II.mp3
- -04- Knots.mp3
- 08 - I Spy.mp3
- captain beefheart - trout mask replica - 05 - hair pie- bake .mp3
- Oblivion_khaos.mp3
- The Dear Hunter - 06 - A Night on the Town.mp3
- 07. Incantation (Part 1).mp3
- 1 robert wyatt SEA SONG.mp3
- 10 Peace To The Mountain.mp3
- 17-Toxicological Whispering.mp3
- 08_-_shadow_of_the_hierophant_320_lame_cbr.mp3
- 04-Stranger In Your Soul_Resampled.mp3
- 02 - Desert Girl.mp3
- 06-Happy Nightmare (Mescaline).mp3
- 07 A Louse Is Not A Home.mp3
- 01 - 2112.mp3
- Dark Light - Sinkin' Deep.mp3
- 101-marillion-the_invisible_man-sns.mp3
- Time Is Monet.mp3
- De-Loused In The Comatorium-The Mars Volta-07-Cicatriz ESP.mp3

Misclassified non-prog as prog songs

- Life's Greatest Fool.mp3
- Bauhaus - Mask - 02.mp3
- 01 the american metaphysical circus.mp3
- 08 - Supersonic Rocket Ship.mp3
- 05-miles_runs_the_voodoo_down_320_lame_cbr.mp3
- Sweet Blindness.mp3
- 03 Sweet Talkin' Woman.mp3
- 10 I.mp3

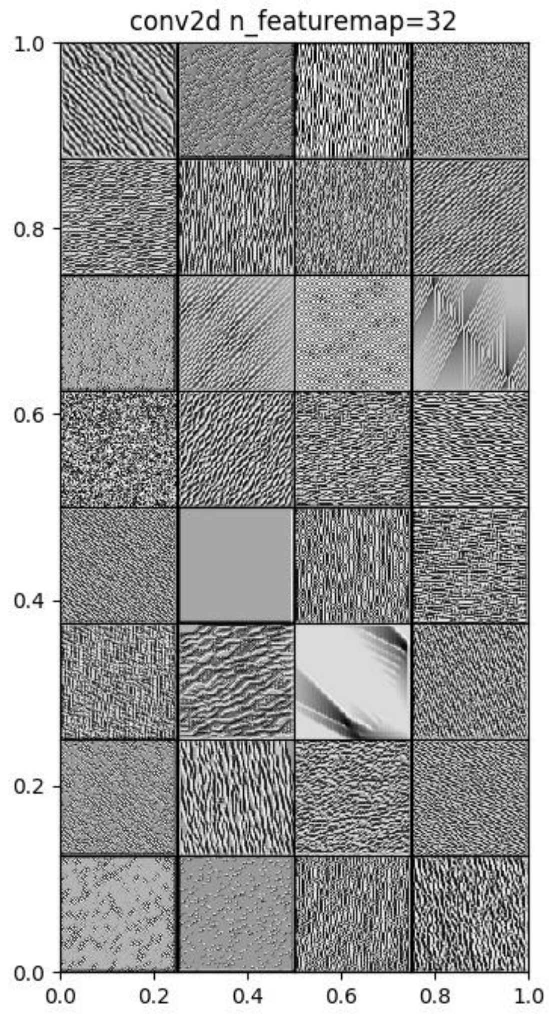
- 03 Katy Song.mp3
- 08-oneohtrix_point_never-computer_vision-kouala.mp3
- 05 - In Your Eyes.mp3
- 02 That's All.mp3
- 06 - Merge.mp3
- 05 - Hey Lawdy Mama.mp3
- 05-Territories.mp3
- 22 - Dear Mr. Fantasy.mp3
- Birthday.mp3
- Fade Away (And Radiate).mp3
- 01 - Highway Star.mp3
- Radiohead - Spectre.mp3
- 01_Mind Ecology.mp3
- 08 - When the Levee Breaks.mp3
- 01 - Heat Of The Moment.mp3
- 07 - Signals (Orchestrion Sketch).mp3
- 09. Tel Aviv.mp3
- 02. None Of Them Knew They Were Robots.mp3

5. Interpret the results

Analysis of misclassified songs
We listen the misclassified songs and consider threre are two main characteristics that caused these songs misclassified. First, the genre is ambiguous. For example, the “Turn of the Century” sounds slow, gentle, and smooth, whose genre is not as strong as the typical prog songs. Second, single song has mixture of different genres. This can cause problem with the majority voting method.

Visualization of kernels
To understand what exactly the CNN model has learned, we visualize the kernels of each convolution layers (Fig. 2 - Fig. 8).

From the first and second layers, we can observe that the kernels basically learn the textures of the Mel-spectrogram. These textures tend to be high frequency and most of them are used to represent the directions



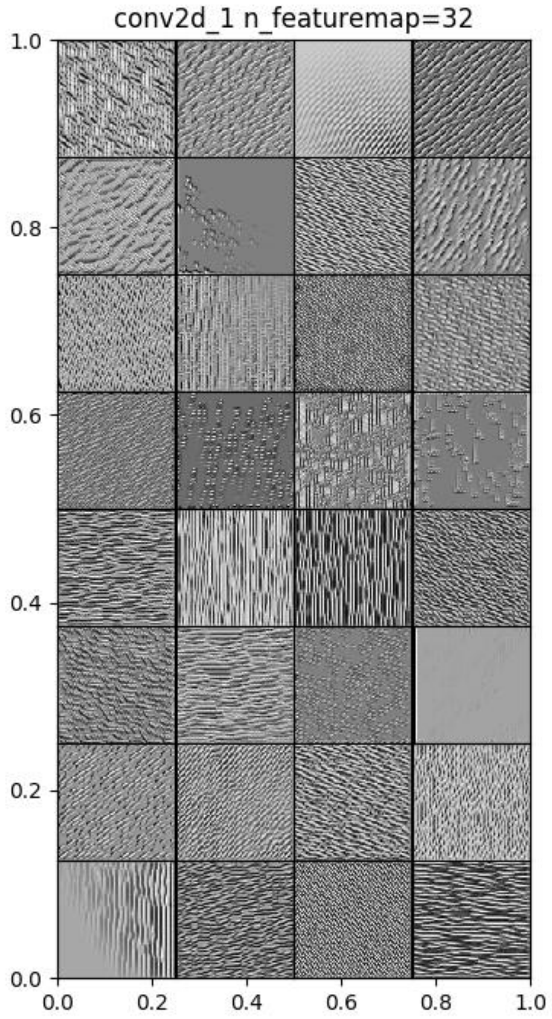


Fig. 2. First convolution layer (#Filter32)

Fig. 3. Second convolution layer (#Filter32)

In the third and fourth layer, some more complex kernels have shown up. These kernels look like the combination of kernels in previous layers with 32 filters. The new filters comprise vertical and horizontal directional texture from previous kernels. This proves that the third convolution layer basically can learn the new features from the features learned from previous layer, the name of "Deep Learning" is kind of making sense.

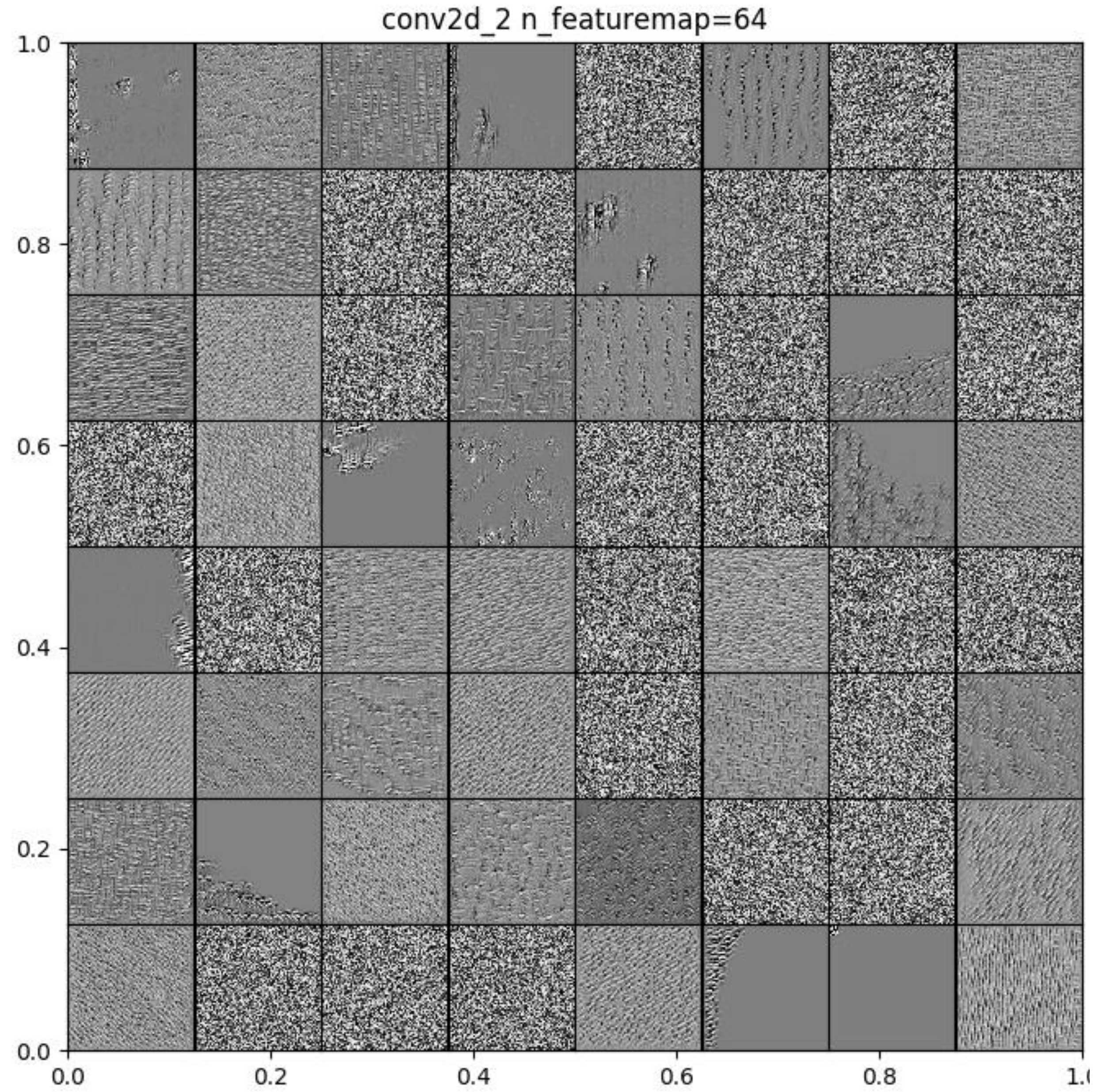


Fig. 4. Third convolution layer (#Filter64)

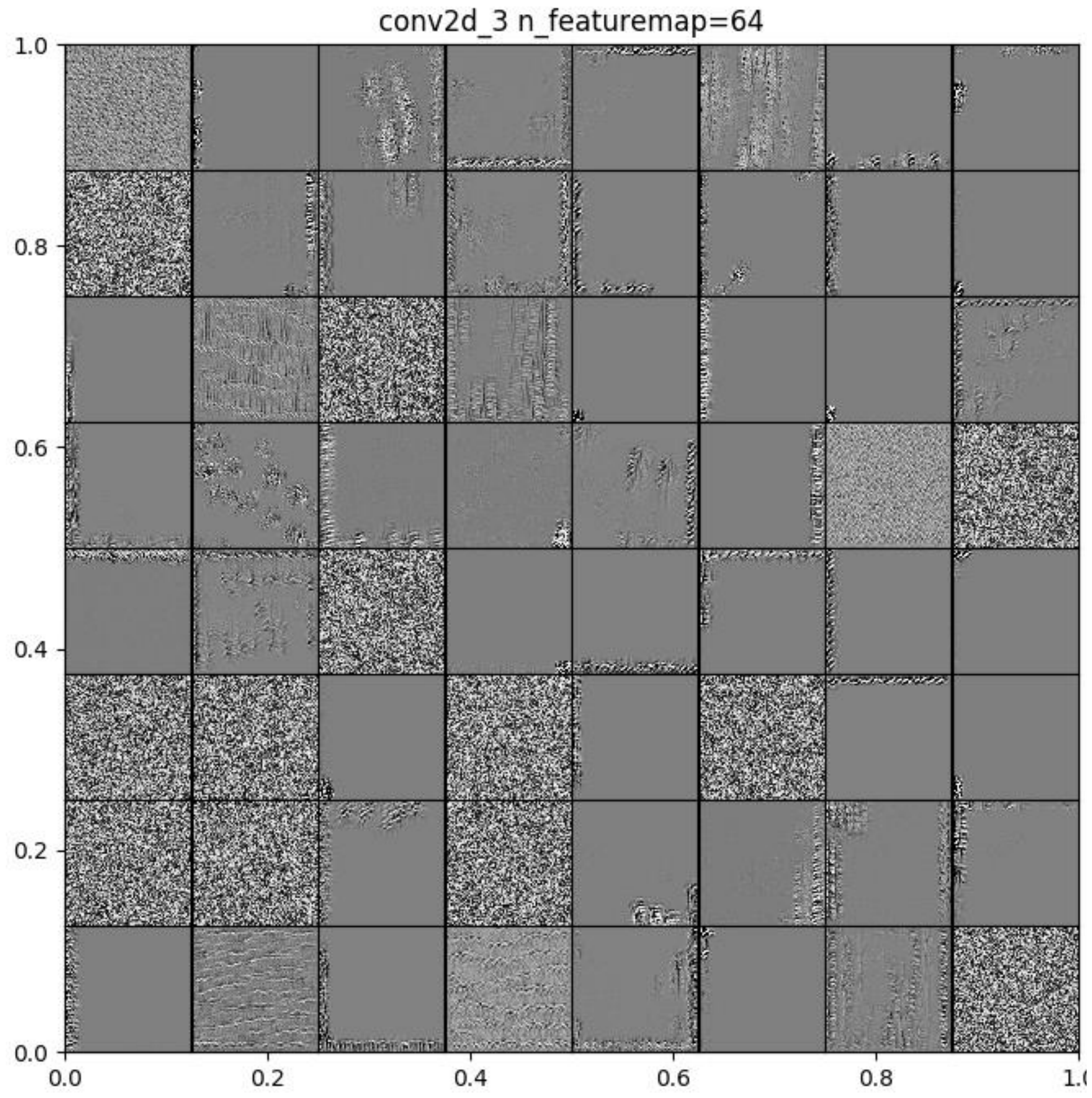


Fig. 5. Fourth convolution layer (#Filter64)

In the fifth and sixth layers, we can see more complex kernels. More interestingly, these kernels somehow look like the Mel-spectrogram. We believe that the kernel looking like Mel-spectrogram indicates that our CNN network has the ability to distinguish the songs with different genres.

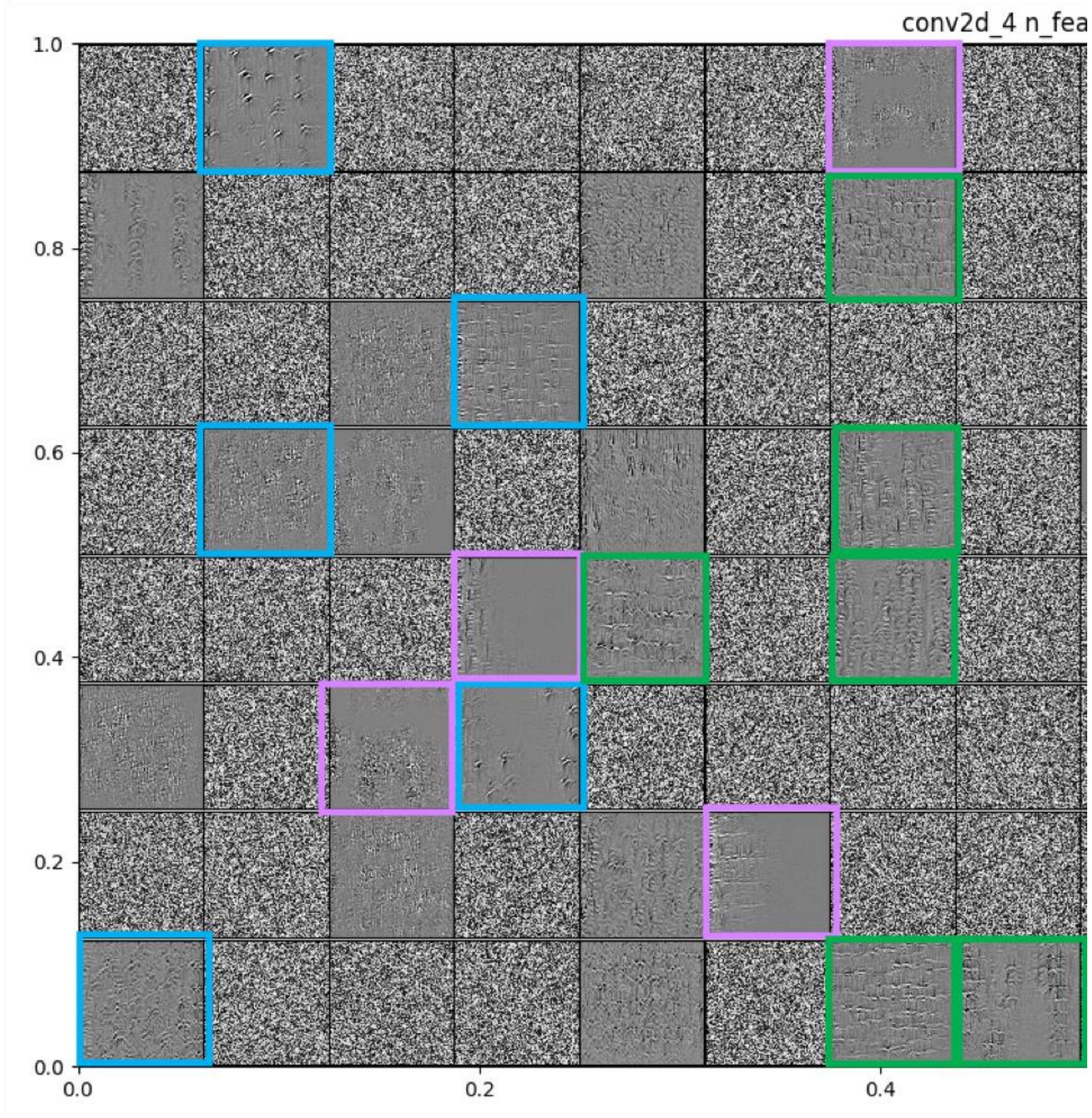


Fig. 6. Fifth convolution layer - 1 (#Filter128)

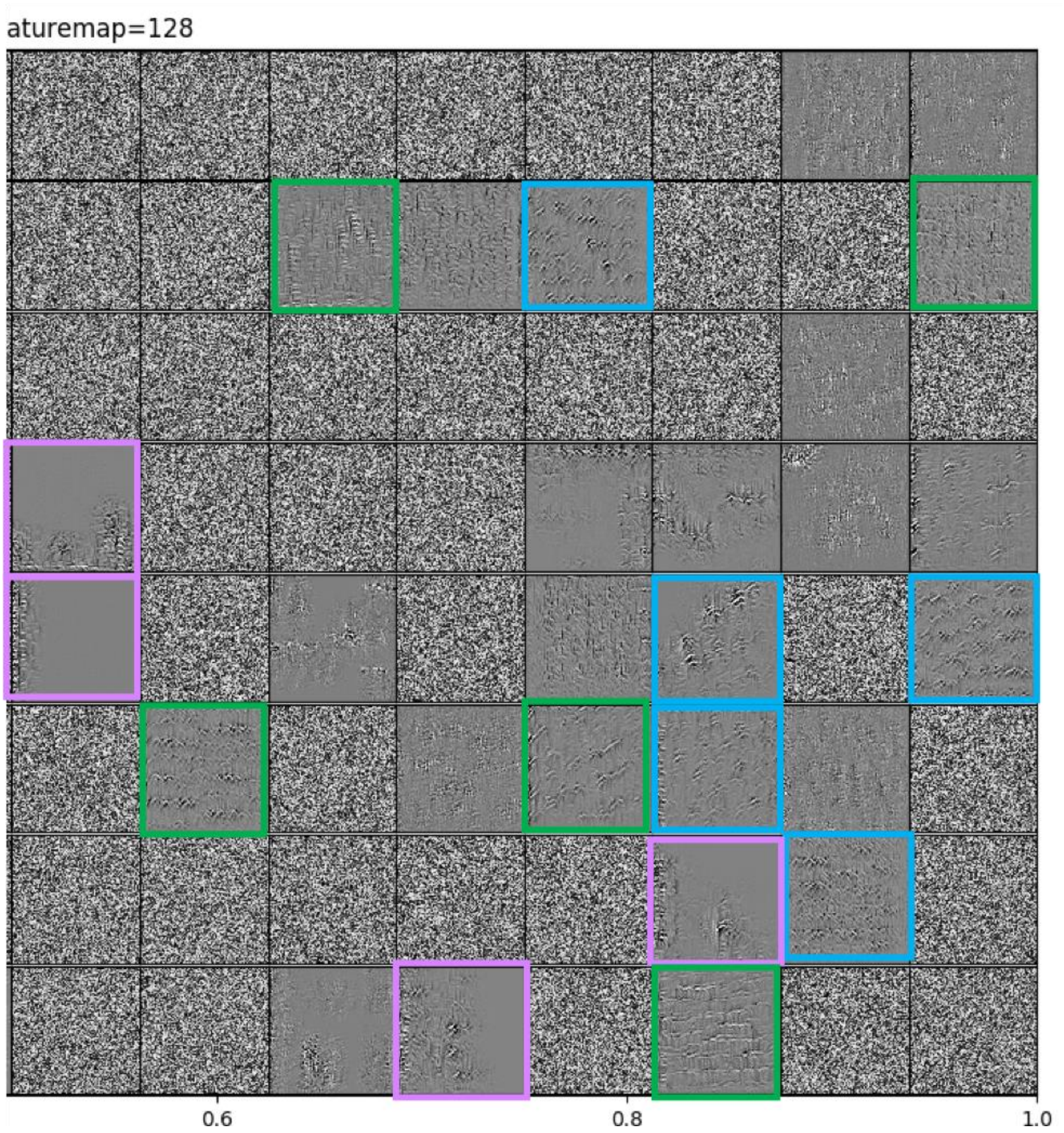


Fig. 7. Fifth convolution layer - 2 (#Filter128)

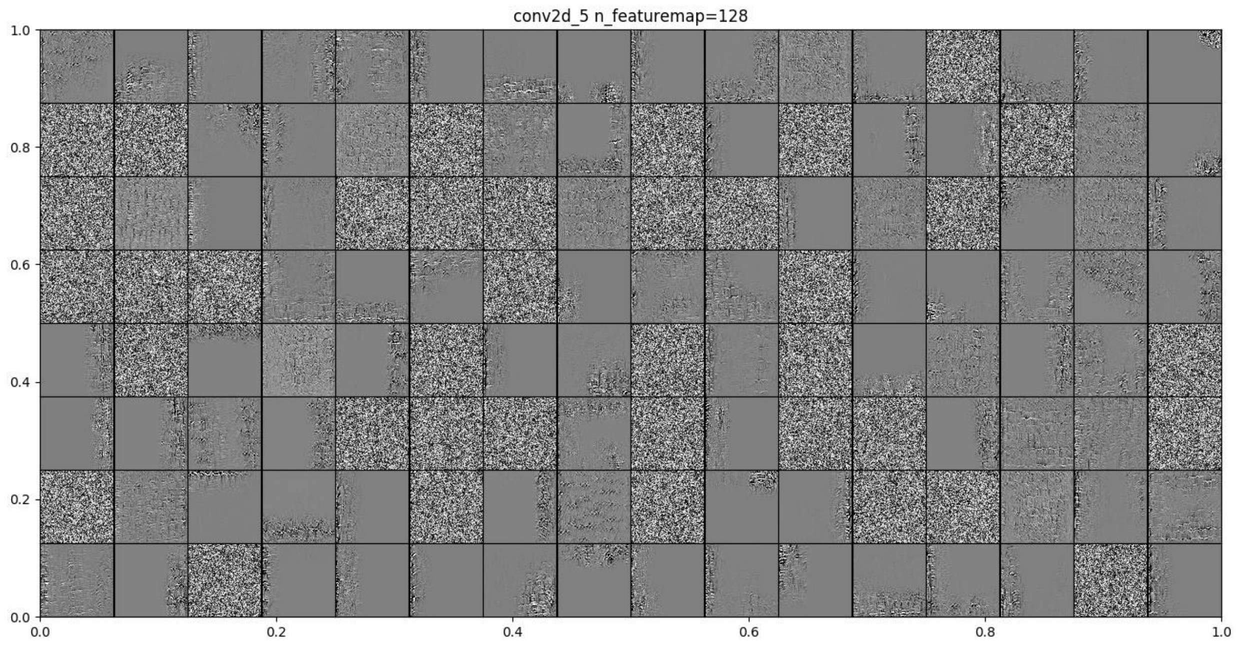


Fig. 8. Sixth convolution layer (#Filter128)

Relationship between patterns and kernels

In this part, we try to take a step forward to see what type of the genres the CNN has learned. The following three Mel-spectrograms are extracted from prog rock music, which are "In the flesh", "Our days are numbered", and "Road of bones" at 4:00.

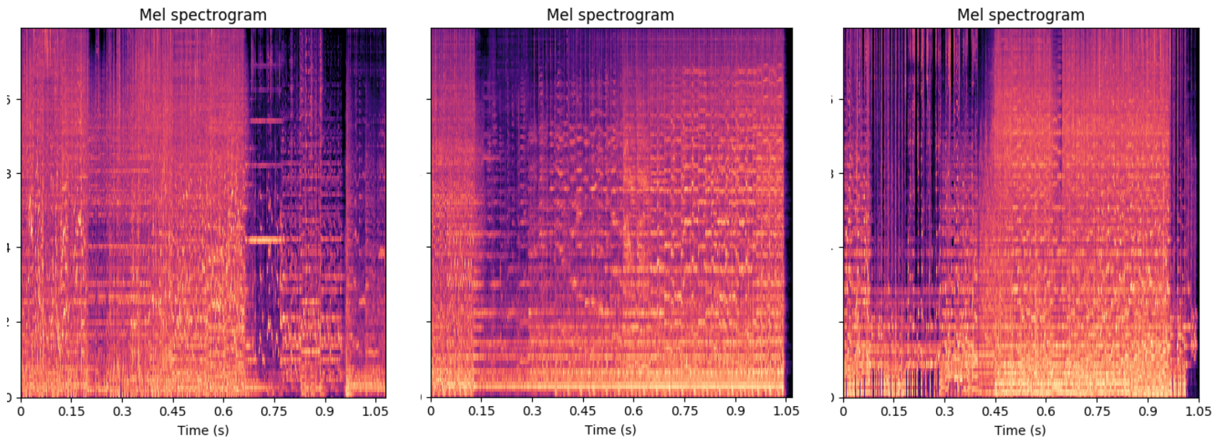


Fig. 9. Mel-spectrograms of segments of prog rock songs

Also, take a look at the Fig. 10. This figure [x] introduces the basic looks of Mel-spectrograms for music with different type of genres.

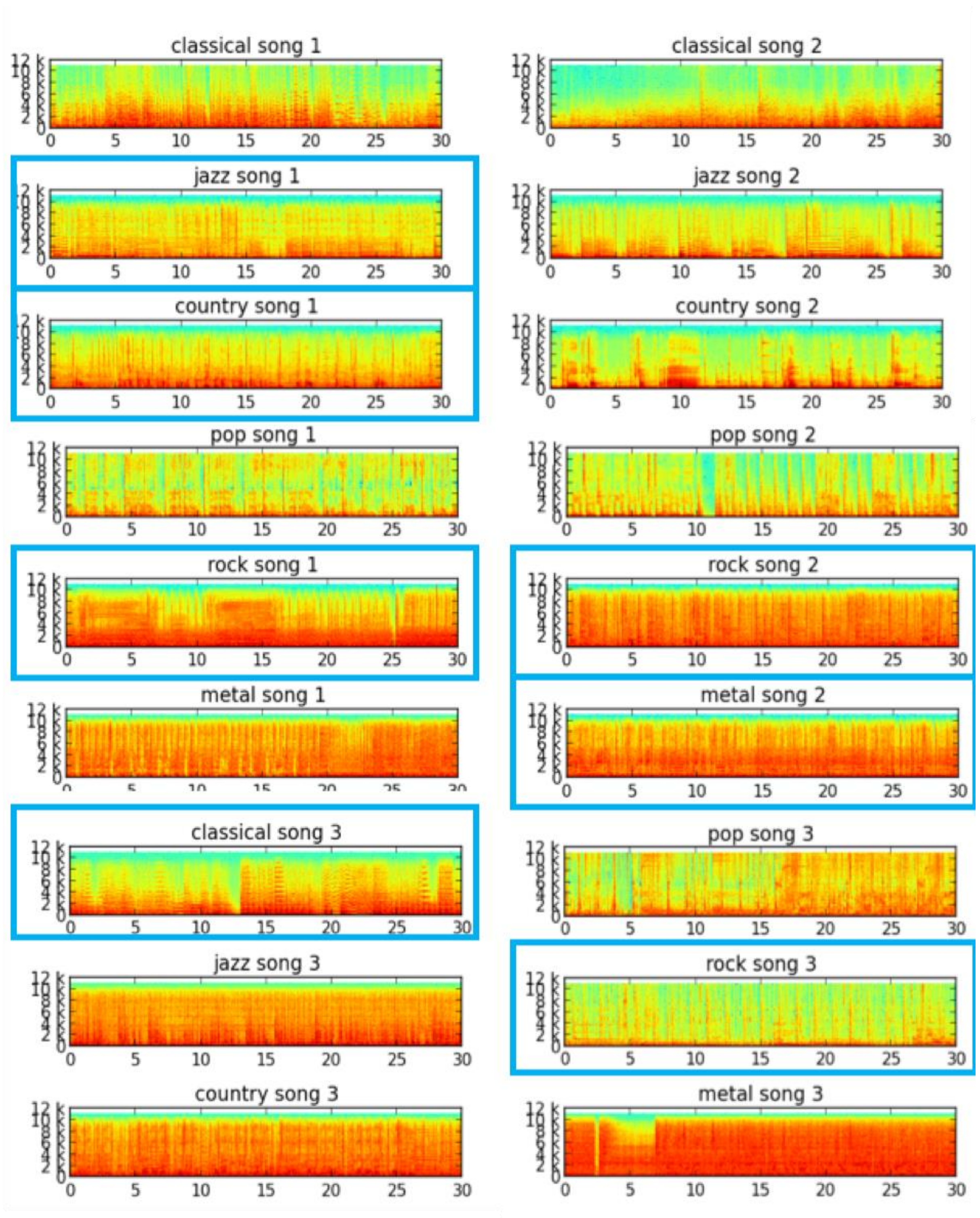


Fig. 10. Mel-spectrograms for different genres.

We can see that the Mel-spectrograms for prog rock is similar to the combination of the jazz, country, rock, and weaker metal genres.

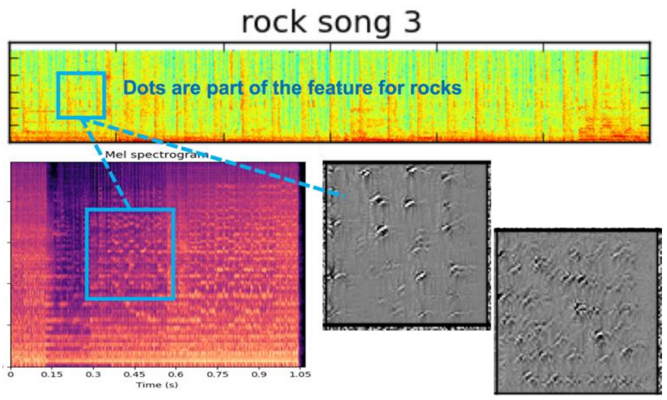


Fig. 11. Dot kernels

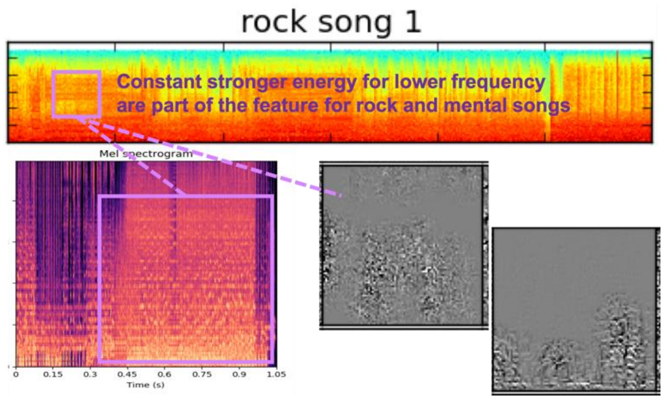


Fig. 12. Low frequency kernels

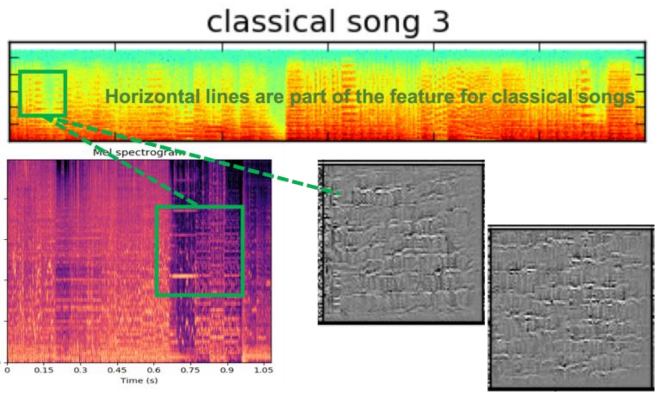


Fig. 13. Horizontal line kernels

Then, we observe that our kernels in convolution layer has learned some genres. The kernels learned the “dot” feature from the rock songs as shown in Fig. 11. The short horizontal lines commonly show up in classical song are also learned (Fig. 13). Finally, the kernel also understand that rock and metal songs usually have stronger energy in low frequency domain (Fig. 12).

Due to the limited space of the article, we only enlarge two represented kernels per feature. The rest of the kernels are marked in Fig. 6 and Fig. 7, where kernels with dots features are highlighted with blue bounding boxes, horizonal line features are with green bounding boxes, and the lower frequency are with purple bounding boxes.

Possible ways to make an improvement

We can notice that lots of kernels in fourth and sixth layers are blank; namely, they learned nothing during the training. These layers are all the “second” layers with same number of kernels with its previous layers. This tells us that except for the beginning layers, there is no need to duplicate the convolution layers with same size. In this way, the model can be more light weight and be harder to overfit.

Another way to improve the performance may be extending the window size. It is possible that the difference between the prog and non-prog rock songs is hard to tell in only one second. If the unique feature for prog rock is switching genres in 30 seconds (maybe 15 seconds for rock and 15 seconds for jazz), it is almost impossible for our kernel to learn this feature limited to the small window size.

From the misclassification song, we know that some prog rock really does not sound like prog rock, and some non-prog rock sounds really like prog rock. As mentioned in class, it might be a good idea to spend some time to label the pattern with fraction number, the stronger prog rock, the higher number we label it, instead of only 0 or 1.

We find that when the number of epochs is too high the model is always overfitting to the non-prog songs. This might be caused by the unbalance number of two class of training patterns, where non-prog patterns is about ¾ more than prog patterns. Therefore, making the number of patterns for two classes equal can elevate the accuracy.

Reference

[1] http://www.image-net.org/
[2] https://deepmind.com/research/alphago/
[3] Han, Y., Kim, J., Lee, K., Han, Y., Kim, J., & Lee, K. (2017). Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 25(1), 208-221.
[4] https://colab.research.google.com/notebooks/welcome.ipynb#recent=true

Report: Multilayer Perceptron model

I. Techniques

1. Extracting features from songs

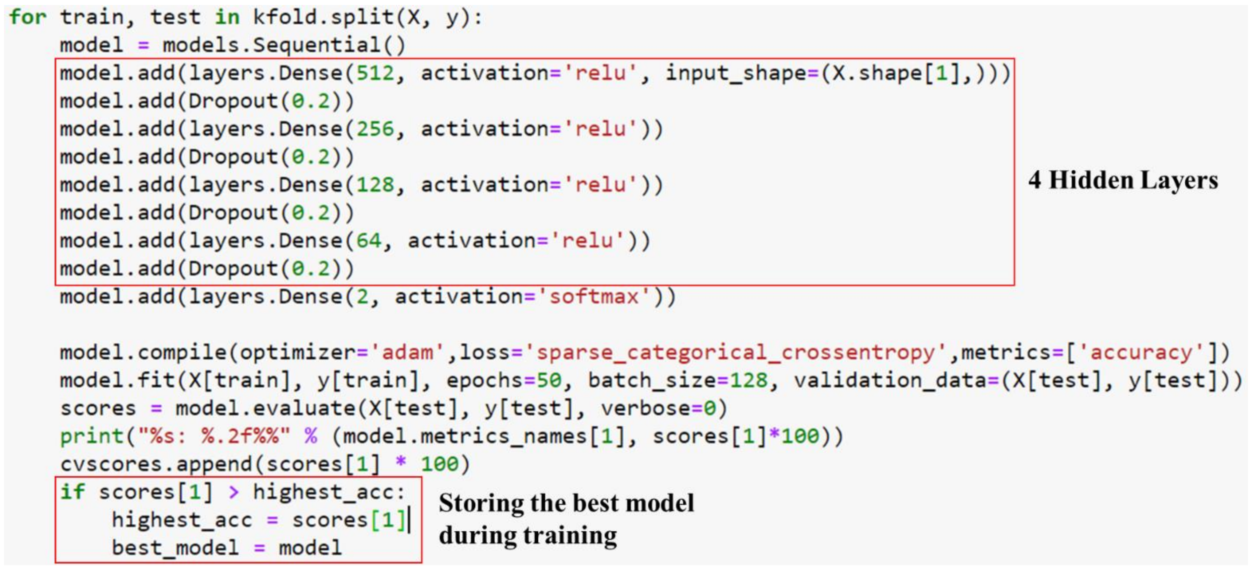
Similar to the method used in LSTM for extracting features, we extract the characteristics (features) of each song and convert each song into an array. To extract features, we use Librosa library, a Python module to analyze audio signals. Below is the list of features.

- Mel-frequency ceptral coefficients (MFCCs)
- Chroma_stft
- Chroma_cqt
- Chroma_cens
- Melspectrogram
- Rms
- Rmse
- Spectral centroid
- Spectral_bandwidth
- Spectral_contrast
- Spectral_flatness
- Spectral_rolloff
- Poly_features
- Tonnetz
- Zero_crossing_Rate

We extract 423 features from each song, including mean and variance of each feature, and thus each song can be regarded as a vector in R^{423} space. These converted vectors will be used as inputs of our MLP model for binary classification.

2. Training and Testing

Since we have two classes: Prog and Nonprog, we use Multilayer Perceptron (MLP) model consisting of 4 hidden layers with the Keras library for binary classification. Actually, the model which we really use is MLP for multiclass classification with softmax for activation function in output layer and categorical crossentropy for loss function. But by setting the number of neurons in output layer to 2, our model becomes 2-class classifier. Compared to MLP for binary classification with sigmoid for activation function in output layer and binary crossentropy for loss function, our model tends to predict validation set more accurately (at about 2~3% averagely). Below is the training part of our model.



```
for train, test in kfold.split(X, y):
    model = models.Sequential()
    model.add(layers.Dense(512, activation='relu', input_shape=(X.shape[1],)))
    model.add(Dropout(0.2))
    model.add(layers.Dense(256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(layers.Dense(128, activation='relu'))
    model.add(Dropout(0.2))
    model.add(layers.Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(layers.Dense(2, activation='softmax'))

    model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
    model.fit(X[train], y[train], epochs=50, batch_size=128, validation_data=(X[test], y[test]))
    scores = model.evaluate(X[test], y[test], verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    if scores[1] > highest_acc:
        highest_acc = scores[1]
        best_model = model
```

During training, we use cross-validation on training set, which means that training set itself is split into training set and validation set randomly and after training model with that training set, we measure performance of model with that validation set. After performing cross-validation 15 times, we store the best model and perform test on validation set and test set with the best model. To avoid overfitting, we apply Dropout to the output of each hidden layer. Dropout is a form of regularization that randomly drops some proportion of the nodes that feed into a fully connected layer. Dropout alleviates the possibility that our neural network may depend excessively on the features by some particular powerful neurons, which might represent a quirk of training set. Thus, it helps to prevent our model from overfitting.

II. Results

1. Cross-validation of the Training Set

Below is the confusion matrix of the result of the best model obtained during cross-validation.

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	73/73	0/73
Actual: Nonprog	1/302	301/302

In overall, the accuracy on prog song is 100%, on nonprog is 99%, and overall accuracy is 99.7%.

2. Measuring the Accuracy on Validation Set

	Predict: Prog	**Predict: Nonprog**
Actual: Prog	10/21	11/21
Actual: Nonprog	12/71	59/71

In overall, the accuracy on prog song is 47.6%, on nonprog is 83%, and overall accuracy is 75%.

3. Measuring the Accuracy on Test Set

	Predict: Prog	Predict: Nonprog
Actual: Prog	34/101	67/101
Actual: Nonprog	6/101	95/101

In overall, the accuracy on prog song is 33.6%, on nonprog is 94%, and overall accuracy is 63.8%.

4. Measuring the Accuracy on Test Set (Using Validation Set)
Here, we use both training set and validation set for training our model and measure accuracy on test set. By using validation set for training, we can get better result of accuracy on test set.

	Predict: Prog	Predict: Nonprog
Actual: Prog	38/101	63/101
Actual: Nonprog	1/101	100/101

In overall, the accuracy on prog song is 37.6%, on nonprog is 99%, and overall accuracy is 68.3%.

5. Explanation
The model is bias toward nonprog song (tends to predict a song is nonprog). The main reason is that we do not segment a song in this model, instead we extract features and calculated their means and variance for a whole song. Thus, given a prog song, their "prog" attribute is shadowed by their majority part that may not sound like a prog. We admitted that the model can be improved by instead of extracting features of whole songs, we should segment it into small parts and extract features of each segment. Then the prediction should be made based on majority vote of mean score as in LSTM.