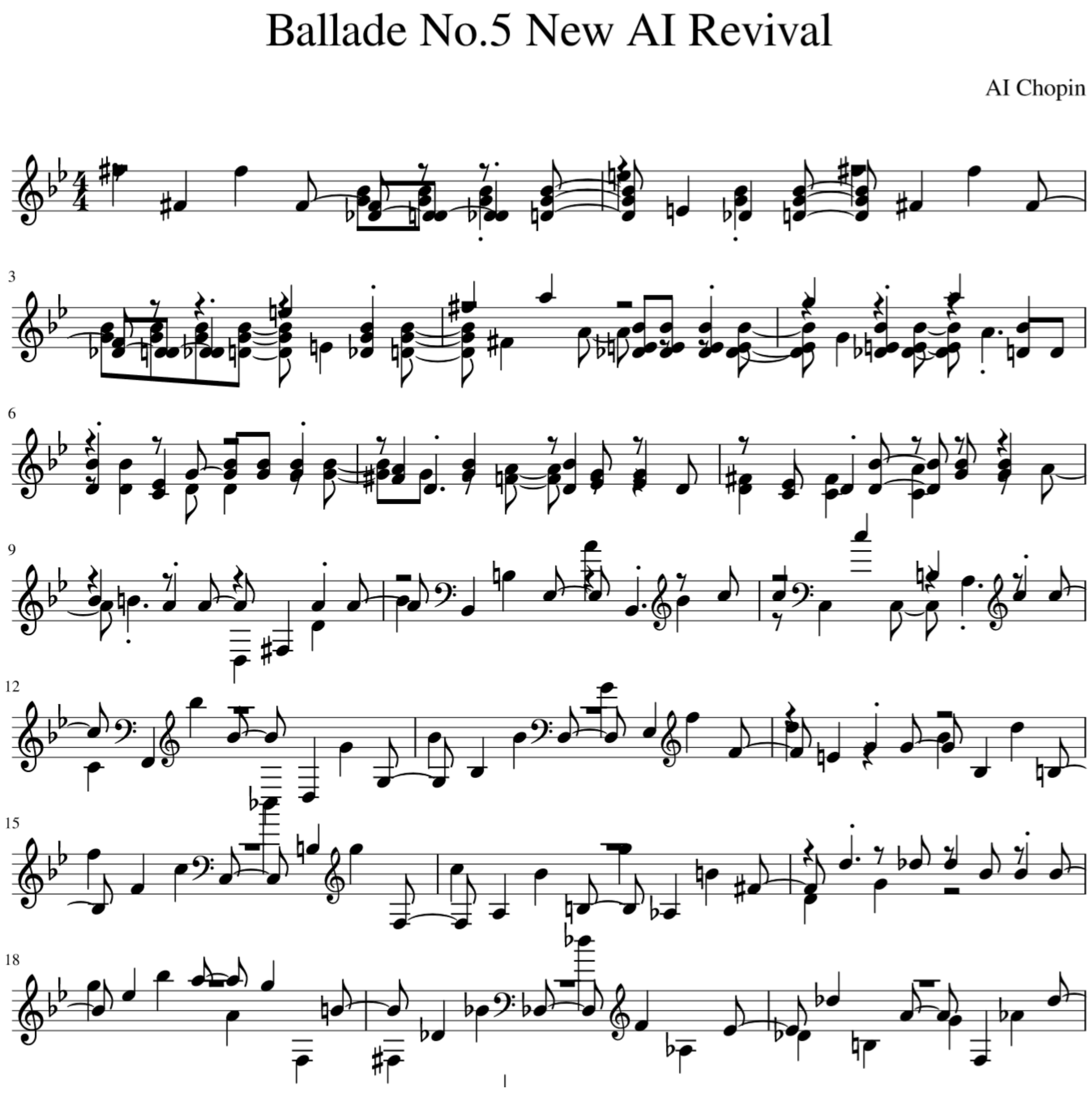
Chopin Music Generation and Classification

Yijian Zong
UCSD
yizong@ucsd.edu



Abstract

In this research paper I will primarily focus on sharing a method to generate and classify Chopin's Music through the deep learning arena of Machine Learning, using convolutional neural networks (CNN), Recurrent Neural Networks(RNN), and Long Short Term Memory(LSTM). I evaluate the validity of architectures through small epochs and benchmark models, and then finetune the structure and add more epochs to produce optimal performance. In the music generation task, I achieved great results through 12 hours of training with a loss of 0.5684. In the classification task, through the usage of Mfcc melspectograms as input to the convolutional neural network, I present the best results and the architecture with which I managed to obtain those results. The CNN achieved a training accuracy of 0.8980 and validation accuracy of 0.7195, a much better model than the Adaboost model based on Transfer Model. Future work may include RNN and CNN visualization and also experimenting with different music analysis metrics and more architectures to improve the accuracy.

Introduction & Motivation

While there is immense amount of research on music recommendation and genre classification, music generation and classification within each genre have been underexplored. Such circumstance could be attributed to the intrinsic difficulty and seemingly non-necessity of the subject. However, a collective tackle on such problem could lead to the future of Natural Language Processing. Endowing Artificial Intelligence with the capability to compose and differentiate between the subtle styles of each artists could lead to the singularity in the art world. This is an exciting mission to me and the entire human race. Such adventurous undertaking could lead to more inspirational thoughts for the interaction between humanity and technology. The neural network learns the features of a piece composed by an artist that make it more likely or less likely to belong to one genre or another. Then it's able to compose in the style of the musician and classify his/her pieces among other.

In this project I will use Convolutional neural networks for the Classification task and Recurrent neural networks with LSTM as building units for the music generations task. CNN is a perfect tool for classification task. In the past decade, we have witnessed a lot of progress in this area until even reaching better than human accuracy for image classification. For example, the MINIST database achieved a remarkable error rate of only 0.21% with CNN. These neural networks consist of a convolutional layer followed by pooling layer. These networks learn to recognize different features of the input and when stacked one after each other, more complex features are learned. Some methods are also introduced as optimization functions. For instance, dropouts are used to avoid overfitting, batch normalization to initialize weights, etc. RNN is a great tool for audio and text generation. These networks are given a sequence of inputs, then generate new outputs with the given weights, and eventually update the weights through backpropagation to improve accuracy. With LSTM cells, RNN can have the ability to go backward more freely and stably with cell state and gates, thereby being more resistant to the vanishing gradient problem. In this project I'll be using mainly convolutional layers, max pooling layers, average pooling layers, RNN layers, and LSTM cells.

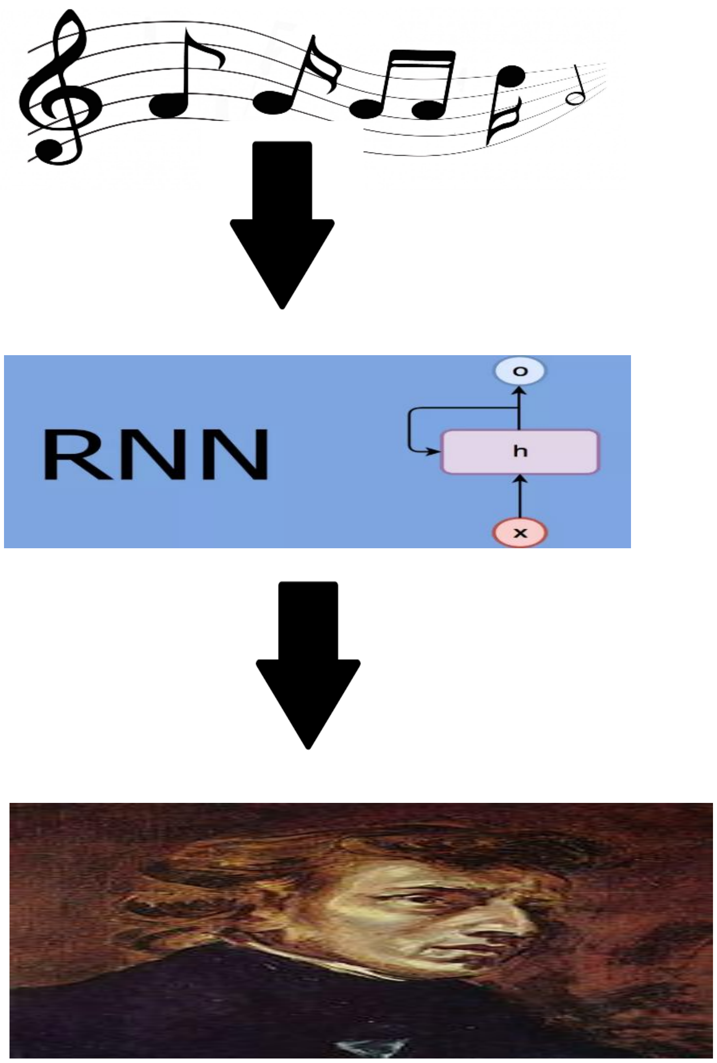


Figure 1. AI composer flow chart

The dataset for training will be divided into two parts: 50 midi files of Chopin and 208 Chopin mp3 files and 201 Other composers mp3 files. The former will be used for generation task since the parsing software Mus21 only support midi files for now. The latter will be used for the Chopin classifier training. The labels for the classification task will be “Chopin” and “Other”.

Related work

Some related works on using neural networks have been published for music genre classification and generation.

Miguel Flores Ruiz de Eguino [1] used GTZAN and MagnaTagATune dataset for genre classification. He experimented different structures such as CNN, Resnet, and achieved good results.

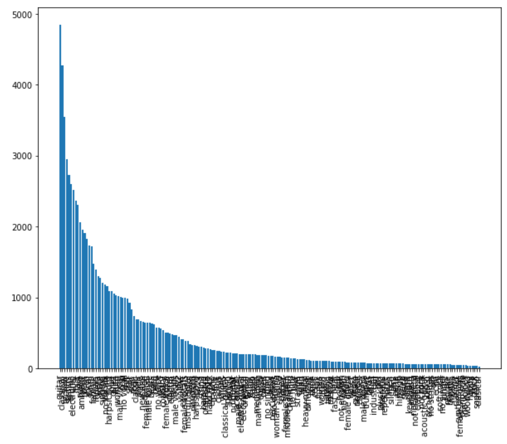


Figure 2. MagnaTagATune tag distribution

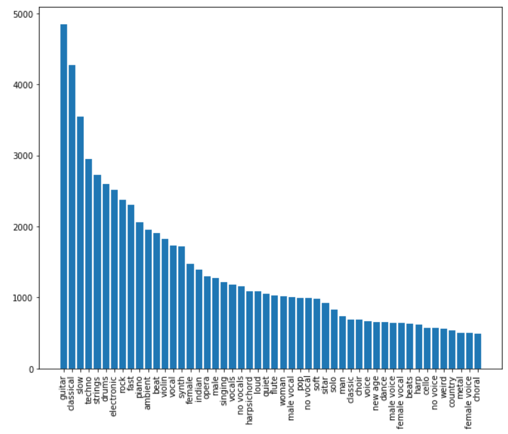


Figure 3. MagnaTagATune tag distribution for the top 50 tags

Keunwoo Choi [2] offered a method for transfer learning using a pre-trained CNN music tagging model. In his paper, he tackled two specific tasks: Ballroom music classification and GTZAN music classification.

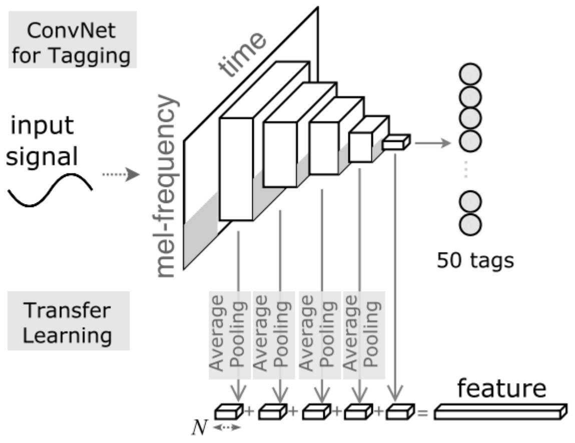


Figure 4. Choi's transfer model

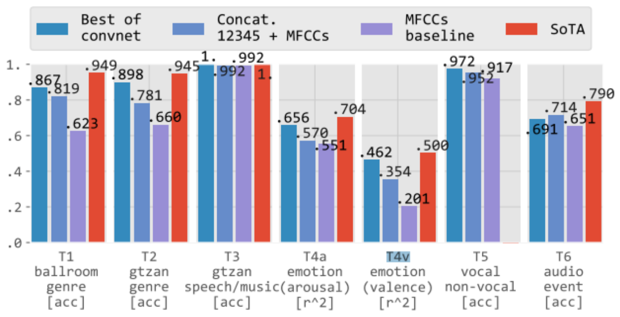


Figure 5. Choi's model performance

Dataset and features

In this paper, I have used the publicly published midi files from Classical Piano Page for the AI composer training and Chopin mp3 files bought by my account. I downloaded 50 midi files and mp3 files. The classification tasks take in a balanced dataset, 208 for Chopin, 201 for other composers.

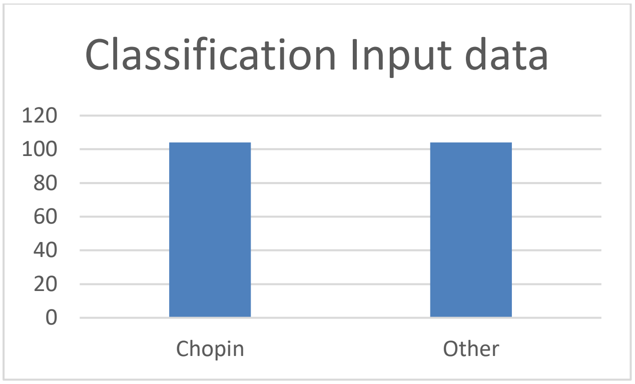


Figure 6. Classification Dataset

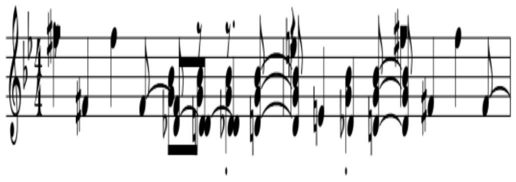
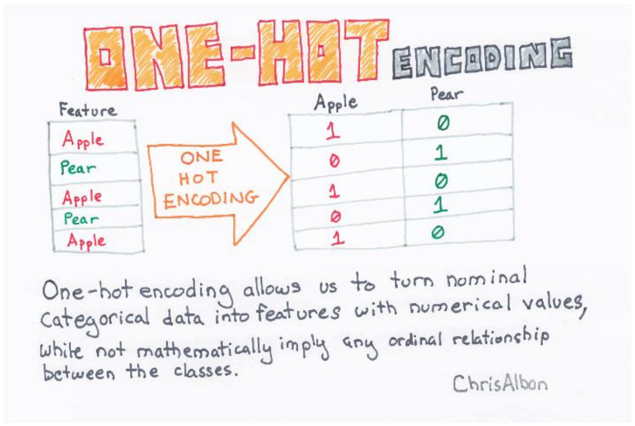


Figure 7. data visualization

Originally I was planning to directly use mp3 files for training. However, the music parsing software Mus21 doesn't parse audio but only song metadata, aka midi files.

In the classification, I used one-hot encoding for binary classification, with 1 being Chopin and 0 being other.



Feature
Apple
Pear
Apple
Pear
Apple

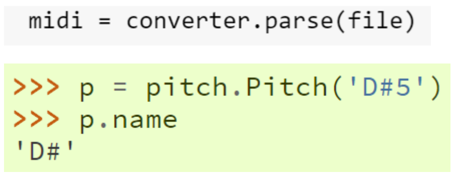
Apple	Pear
1	0
0	1
1	0
0	1
1	0

One-hot encoding allows us to turn nominal categorical data into features with numerical values, while not mathematically imply any ordinal relationship between the classes.

Figure 8. one-hot encoding visualization

Data Preprocessing

For the RNN task, we preprocess the data by using a software called Music21 to get notes of the midi file.

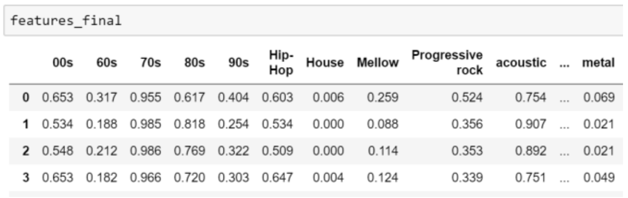


```
midi = converter.parse(file)
```

```
>>> p = pitch.Pitch('D#5')
>>> p.name
'D#'
```

Figure 9. sample of parsing midi and getting notes

For the CNN task, we firstly attempted with pandas dataset using the transfer model published by Keunwoo Choi, which turned out to be ineffective way.



```
features_final
```

	00s	60s	70s	80s	90s	Hip-Hop	House	Mellow	Progressive rock	acoustic	...	metal
0	0.653	0.317	0.955	0.617	0.404	0.603	0.006	0.259	0.524	0.754	...	0.069
1	0.534	0.188	0.985	0.818	0.254	0.534	0.000	0.088	0.356	0.907	...	0.021
2	0.548	0.212	0.986	0.769	0.322	0.509	0.000	0.114	0.353	0.892	...	0.021
3	0.653	0.182	0.966	0.720	0.303	0.647	0.004	0.124	0.339	0.751	...	0.049

Figure 10. sample of input for Adaboost classifier

Then I started using MFCC specotgrams directly, which produced a much better result.

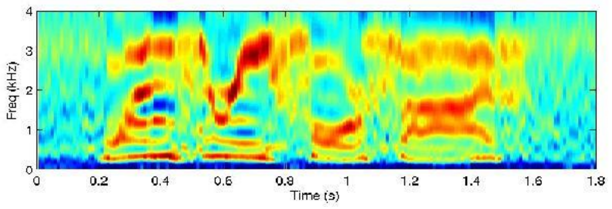


Figure 11. sample of input with MFCC specotgrams

Metrics

For RNN, cross entropy is a decent metrics to be used on the test set to assess how accurate the model is predicting the test data in the realm of natural language processing:

$$H(T,q) = -\sum_{i=1}^{N} \frac{1}{N} \log_2 q(x_i)$$

Figure 12. Cross entropy calculation

This metric was used when evaluating the RNN generator because, through the activation function softmax, the cross-entropy loss function can capture the probabilistic character of the network and efficiently update the weights to perform better.

For CNN, since the dataset is balanced, the most efficient way to measure performance should be accuracy. Accuracy is tailored made for binary classification, taking into account both true positive and true negatives with equal weight.

Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$

Figure 12. Accuracy calculation

For the experimental Adaboost model, we use F score along with Accuracy.

For testing the scope problem, I introduce a new metrics called blind accuracy metrics, which introduce completely new random data for testing. Blind accuracy is calculated in the same way as normal accuracy.

RNN Architecture Design

Studies have shown RNN with LSTM as units are effective in the realm of NLP. RNN algorithm consists of several recurrent units, processing sequences of inputs using matrix operations and output RELU as product to hidden layers, then using softmax function to give a probability result.

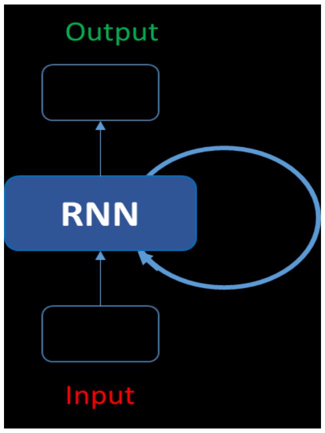


Figure 13. Simple RNN

The LSTM is a very special kind of RNN, which has cell state, gates, and four neural network layers, compared to one of simple RNN.

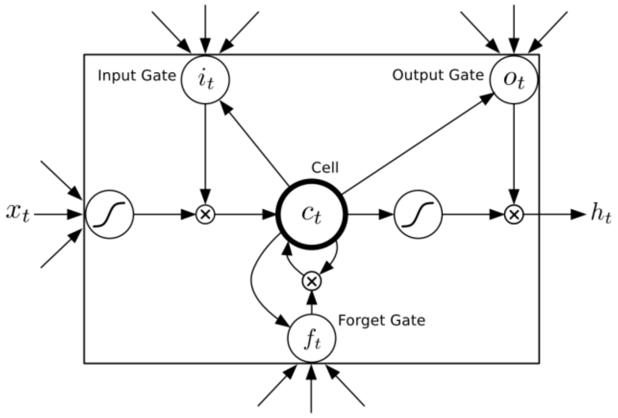


Figure 14. LSTM

Benchmark RNN

For benchmark model, we use RNN with three LSTM layers, one dense layer, and one softmax output layers. The reason is that this can test the validity of the algorithm quickly.

Optimized RNN

For more sophisticated model, we use RNN with three LSTM layers, two dense layer, three dropout layers, and one softmax output layers.

LSTM layers are the most essential units of the architecture. They take in sequences of inputs and use sigmoid layer acting like a “forget gate layer” to decide what information to throw away from the cell state. It looks at inputs, return a number between 0 and 1, with 1 representing “keeping” and 0 “throwing”. Then it decides what new information to store. It uses tanh layer to calculate a vector of new values to consider. Finally it updates the old state into new cell state, filtering out less useful information and output the result.

Dropouts layers are used to prevent overfitting through randomly throwing out some data in different nodes.

Dense layers are used to shaping the layers to give an expected output.

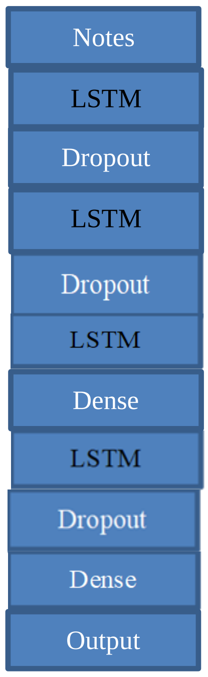


Figure 15. RNN design

Result

After 12 hours training on AWS, the AI composer generated very promising result. While the result from benchmark model is very simple and naïve, with single note repeated all along, the result after final training with optimized structure showed decent musical structure, with base chords and main melody. Music and visualization can be found on my Github.

The training is done using rmsprop optimizer. After training of 60 epochs, the loss was reduced from 4.5977 to 0.4951, which is a remarkable improvement. The sheet of the generated music also show some signs of patterned chords and style of Chopin.

Figure 16. naïve visualization

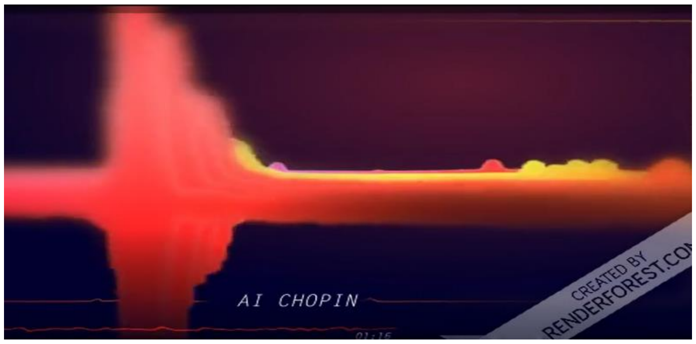
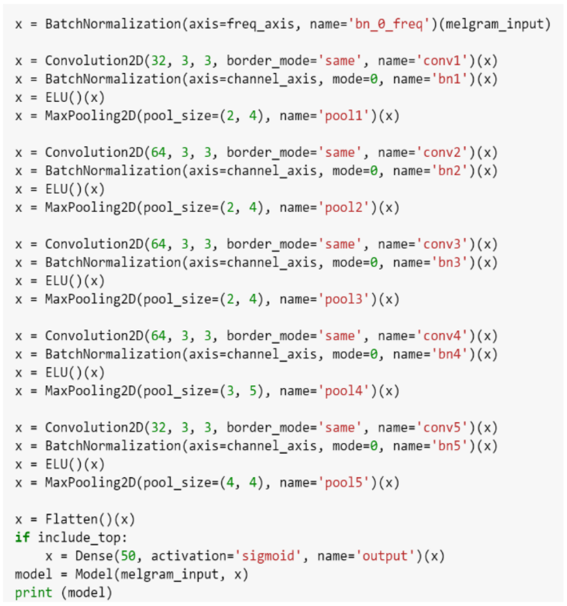


Figure 17. Final result visualization

Chopin Classification

Transfer Model Architecture Design

I used the transfer model published by Keunwoo Choi (Figure 4), with several customer layers to enhance performance. Five Convolutional 2D layers were added, along with Batch Normalization, Elu, and Maxpooling layers. Then data are flattened and sigmoid layer was added at the end. The model uses melgrams for prediction. I then preprocessed the resulting tagging into pandas dataframe and use Adaboost for prediction.

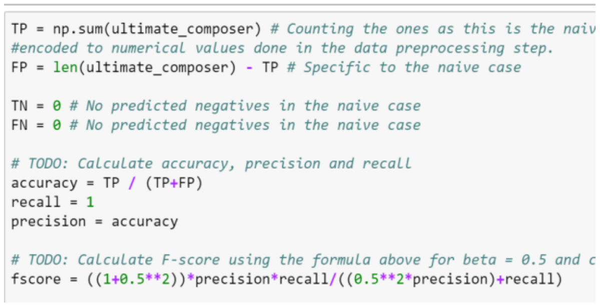


```
x = BatchNormalization(axis=freq_axis, name='bn_0_freq')(melgram_input)

x = Convolution2D(32, 3, 3, border_mode='same', name='conv1')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn1')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(2, 4), name='pool1')(x)

x = Convolution2D(64, 3, 3, border_mode='same', name='conv2')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn2')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(2, 4), name='pool2')(x)

x = Convolution2D(64, 3, 3, border_mode='same', name='conv3')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn3')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(2, 4), name='pool3')(x)

x = Convolution2D(64, 3, 3, border_mode='same', name='conv4')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn4')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(3, 5), name='pool4')(x)

x = Convolution2D(32, 3, 3, border_mode='same', name='conv5')(x)
x = BatchNormalization(axis=channel_axis, mode=0, name='bn5')(x)
x = ELU()(x)
x = MaxPooling2D(pool_size=(4, 4), name='pool5')(x)

x = Flatten()(x)
if include_top:
    x = Dense(50, activation='sigmoid', name='output')(x)
model = Model(melgram_input, x)
print (model)
```

Figure 19. Added layers of transfer model

Benchmark Model

For benchmark model, we simply count all the ones and zeros and use them to calculate accuracy and f-score. The purpose for using this as benchmark model is to show what a model without any intelligence would look like.



```
TP = np.sum(ultimate_composer) # Counting the ones as this is the naiv
#encoded to numerical values done in the data preprocessing step.
FP = len(ultimate_composer) - TP # Specific to the naive case

TN = 0 # No predicted negatives in the naive case
FN = 0 # No predicted negatives in the naive case

# TODO: Calculate accuracy, precision and recall
accuracy = TP / (TP+FP)
recall = 1
precision = accuracy

# TODO: Calculate F-score using the formula above for beta = 0.5 and c
fscore = ((1+0.5**2))*precision*recall/((0.5**2*precision)+recall)
```

Figure 18. Naïve model

Adaboost Architecture Design

Then I used Adaboost to output the final result of prediction. Adaboost works by putting more weight each time on the loss instead of predictions.

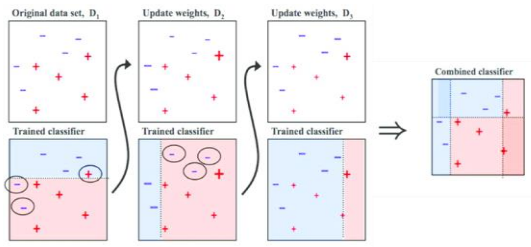
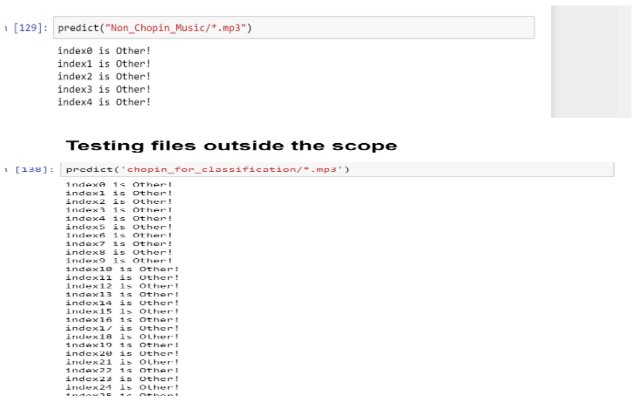


Figure 19. Adaboost algorithm

Result

At the first attempt, I used transfer model and Adaboost for classification. The model caused a scope issue, given that the transfer model itself would predict same results on the same input. The accuracy is 47.62% and the F-score is 49.59% with both optimized and unoptimized model. After using feature extraction, the model reached 61.90% accuracy and 61.98% F-score. However, when we tested the model with external data, it showed that the model output same result. To solve the scope problem, I decided to switch to CNN from transfer learning.



```
[129]: predict("Non_Chopin_Music/*.mp3")

index0 is Other!
index1 is Other!
index2 is Other!
index3 is Other!
index4 is Other!
```

Testing files outside the scope

```
[138]: predict('chopin_for_classification/*.mp3')
index0 is Other!
index1 is Other!
index2 is Other!
index3 is Other!
index4 is Other!
index5 is Other!
index6 is Other!
index7 is Other!
index8 is Other!
index9 is Other!
index10 is Other!
index11 is Other!
index12 is Other!
index13 is Other!
index14 is Other!
index15 is Other!
index16 is Other!
index17 is Other!
index18 is Other!
index19 is Other!
index20 is Other!
index21 is Other!
index22 is Other!
index23 is Other!
index24 is Other!
```

Figure 20. Prediction from first attempt

CNN Architecture Design

CNN is known for its blazing accuracy of image classification. In this study, instead feeding pixels as inputs to the CNN, I use MFCC specotgrams for model training. CNN generally has the following components:

Convolutional layer: this layer will compute the output of neurons connected through the inputs in the form of dot product.

RELU layer: this layer will apply elementwise activation function but doesn't change the size of the volume.

POOL layer: this layer will perform a downsampling operation and decrease the size of the volume.

FC(i.e.fully-connected) layer: This is the layer that compute class scores, in the case Chopin and Other, with the former corresponding to 1, the latter 0.

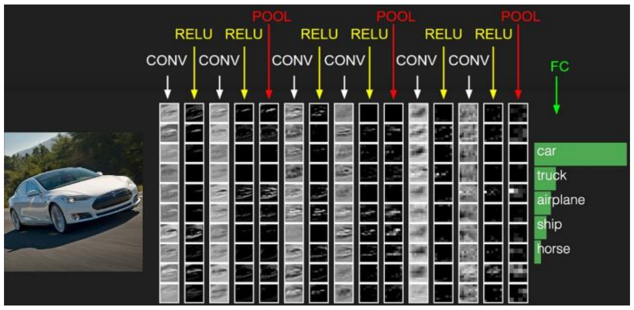


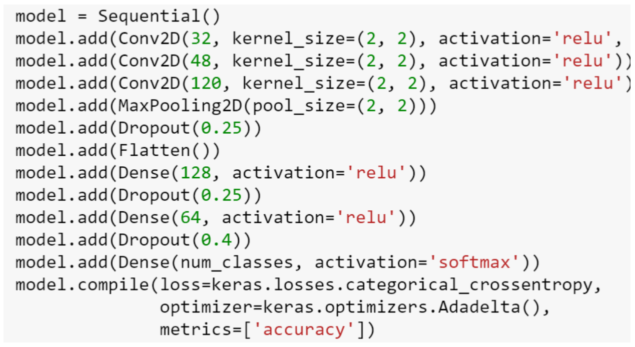
Figure 21. CNN example in image classification

Benchmark CNN

For benchmark model, we only feed CNN with ten mp3 files of Chopin and ten other, along with some random audio files.

Optimized CNN

For the optimized model, we feed into the CNN 208 files for Chopin composition and 201 files for other compositions. A CNN with traditional structure is used, the loss is calculated with categorical cross-entropy, along with accuracy metrics and optimizer Adadelta.



```
model = Sequential()
model.add(Conv2D(32, kernel_size=(2, 2), activation='relu',
model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
model.add(Conv2D(120, kernel_size=(2, 2), activation='relu')
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

Figure 22. CNN classifier design

Result

The classifier is trained much faster than the previous RNN model. After 30 minutes training on AWS, the Chopin Classifier powered by CNN produced a tremendous improvement compared to the Adaboost model, with training accuracy of 0.8980, validation accuracy of 0.7317, and a test accuracy of 0.55556 on the Chopin dataset. The model scores 0.3077 on Liszt dataset, which is expected since two composers have very similar styles and interacted in the same era. The accuracy score on mix composers excluding Chopin is 0.3509. This is a remarkable leap from the previous model and show a promising direction for future finetuning.

Justification

RNN model:

As expected, the finals results are much stronger than the benchmark result since the structure of the model is more sophisticated and the training epochs are much longer. The final solution has lowest loss of 0.4951 before overfitting and the benchmark model gave a loss higher than 5.0. This means if we assume the loss distribution is normal, the optimized model shifted from the very right end to the very left end. As further examined by the generated music, the optimized model produced a well-structured music compared to the single notes repetition in the benchmark model generated file. We can safely conclude that the final solution is significant enough to solve the problem.

CNN Model:

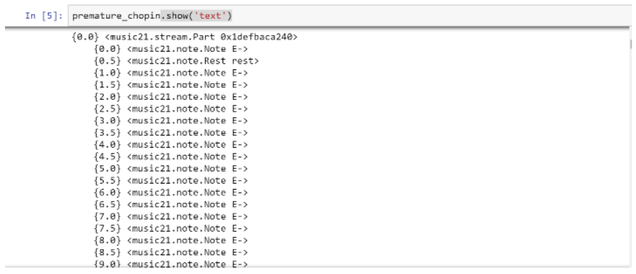
If we compare the CNN model to the benchmark naïve model and even the Adaboost model, the CNN gave a satisfactory validation accuracy of 0.7317, along with training accuracy of 0.8980. If we assume the accuracy distribution is gaussian(normal), the naïve model and Adaboost model would lie on the middle fifty range, while the CNN model would be more than 1 standard deviation from the center. This means even if the CNN model doesn't give an super ideal result, it is a very decent model that distinguishes patterns in music instead of randomly guessing.

Free-form Visualization

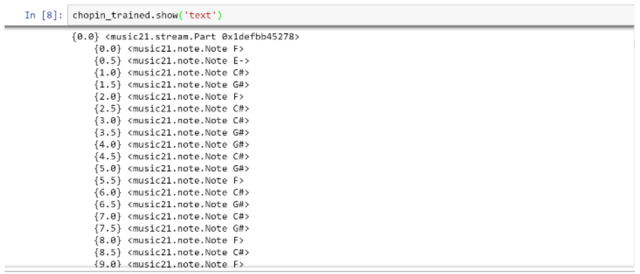
RNN model:

We use two ways to visualize the notes generated to compare what we have been discussing all along – the benchmark model and optimized model.

The first way is raw notes comparison:



```
In [5]: premature_chopin.show('text')

{0.0} <music21.stream.Part 0x1defbaca240>
    {0.0} <music21.note.Note E->
    {0.5} <music21.note.Rest rest>
    {1.0} <music21.note.Note E->
    {1.5} <music21.note.Note E->
    {2.0} <music21.note.Note E->
    {2.5} <music21.note.Note E->
    {3.0} <music21.note.Note E->
    {3.5} <music21.note.Note E->
    {4.0} <music21.note.Note E->
    {4.5} <music21.note.Note E->
    {5.0} <music21.note.Note E->
    {5.5} <music21.note.Note E->
    {6.0} <music21.note.Note E->
    {6.5} <music21.note.Note E->
    {7.0} <music21.note.Note E->
    {7.5} <music21.note.Note E->
    {8.0} <music21.note.Note E->
    {8.5} <music21.note.Note E->
    {9.0} <music21.note.Note E->
```



```
In [8]: chopin_trained.show('text')

{0.0} <music21.stream.Part 0x1defbb45278>
    {0.0} <music21.note.Note F>
    {0.5} <music21.note.Note E->
    {1.0} <music21.note.Note C#>
    {1.5} <music21.note.Note G#>
    {2.0} <music21.note.Note F>
    {2.5} <music21.note.Note C#>
    {3.0} <music21.note.Note C#>
    {3.5} <music21.note.Note G#>
    {4.0} <music21.note.Note G#>
    {4.5} <music21.note.Note C#>
    {5.0} <music21.note.Note G#>
    {5.5} <music21.note.Note F>
    {6.0} <music21.note.Note C#>
    {6.5} <music21.note.Note G#>
    {7.0} <music21.note.Note C#>
    {7.5} <music21.note.Note G#>
    {8.0} <music21.note.Note F>
    {8.5} <music21.note.Note C#>
    {9.0} <music21.note.Note F>
```

Figure 23. Raw notes comparison

As we see, the benchmark models only generate one single note but the optimized model generate different notes. To see whether the notes generated make sense better, we use sheet music to compare:

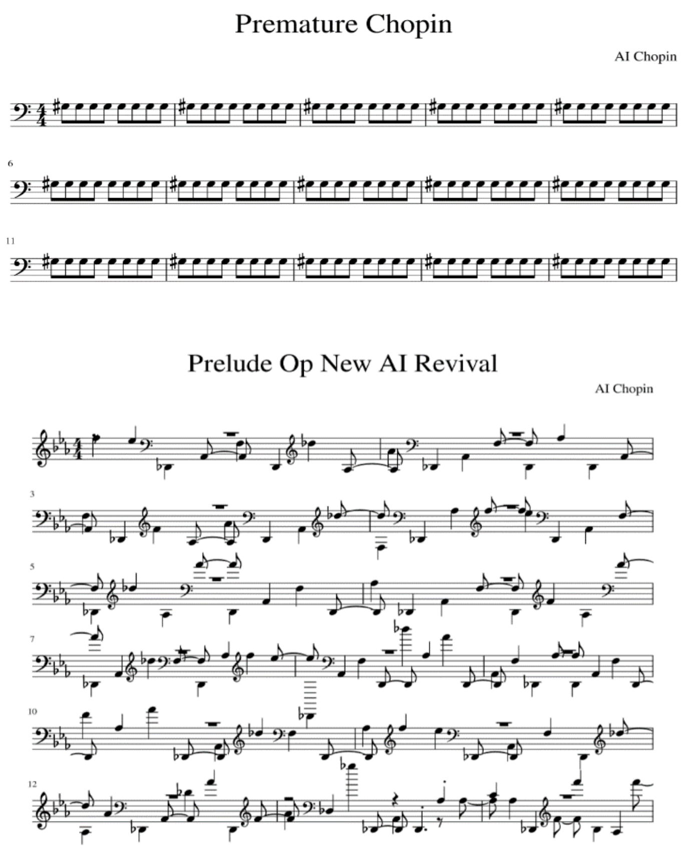


Figure 24. Sheet music comparison

Now we can visualize the difference much more clearly. Even though there are some random notes there in the trained-model, we can see it is much structured, with a main melody and an accompaniment. There are also marks of slur and change to bass and treble, which are commonly used in Chopin's and classical music.

Conclusion and Future work

In the music generation task, I managed to achieve decent accuracy on the datasets I worked with and the AI can compose some beautiful melodies. However, the music generated still showed there is long road that machine learning should walk to achieve the ability of average music composers, let alone Chopin. For future research, we can use melgrams, which are proved to be efficient than MFCC, or chords recognition, to do semi-supervised learning on the generation model for more sophisticated result. In the classification task, I bumped into a scope issue that can only be fixed by switching to a new predicting model. In the new CNN classifier, the model achieved good results and classified many random mp3 files correctly. On average, mp3 will be harder to predict than midi, which is just the music sheet in the form of metadata. Despite this, there are a lot of collective research needed to improve the accuracy and achieve a composer classification result similar to those of image classification. It is even hard for me and other people to differentiate between different composers, since some have very similar styles. Therefore, for future exploration, more professional metrics (chord analysis, harmony analysis, etc.) should be introduced.

References

[1] B. S. Aaron van den Oord, Sander Dieleman. Deep content-based music recommendation. 2013.

[2] M. S. Keunwoo Choi, Gyorgy Fazekas. Automatic tagging using deep convolutional neural networks. 2016.

[3] Miguel Flores Ruiz de Eguino. Deep music genre. 2017