## Ian Edwards CS4390 Deep Learning Project

## Summary

For our Deep Learning project, we chose to implement a Recurrent Neural Network to make a machine that generates music.

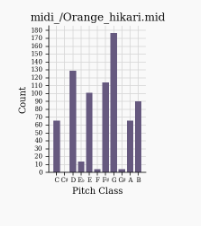
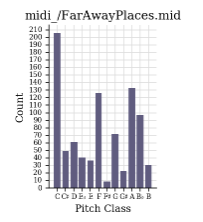
## Grading Criteria Points:

### Effort to curate dataset

1000+ notes, chords, and rhythms were used in the dataset. The notes were obtained by using song files converted and/or downloaded from a .mid or midi format. The first round of training was only done with 6 songs. Which later fluctuated between 24 and 30 songs. Notes were used as data points since the model used them to predict the next sequence whether it be a note or a chord. I personally aimed for songs from my favorite Anime shows and some piano songs that I felt were jazz and relaxing. The program was ran multiple times but I only kept a record of the distinctly generated results.

### Effort to visualize input data

These images represent the number of data points per song used for the training



### Effort to correctly split data into 3 sets

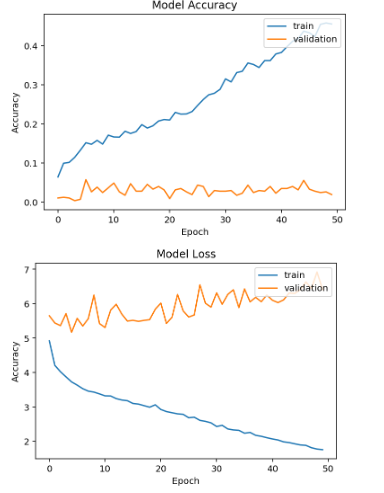
We perform a train to validate split using keras.model.fit validation\_split parameter instantiated with a 0.2 cut, or 80% to 20% data split between train and validate. Since we are using generative modeling we did not need a test data set. This is because generative modeling means building a model that can generate new examples that come from the same distribution set as the training data. This indicates that generative modeling is considered unsupervised learning.

### Effort to design/test neural network architectures -- 9. Effort to study Learning curves

Our model architecture uses four layer types. LSTM, Dropout, Dense, and Activation. From the beginning we started out using generative models to produce new musical examples. We each developed a unique data set to train and validate the model, some of the various architectures used are discussed below. A variety of trainings yielded similar loss, val\_loss, and song results so they’re omitted for brevity.

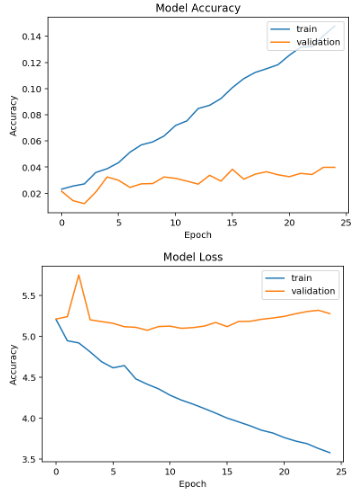
FIRST TRAINING: 6 songs

* model model = Sequential()
* model.add(LSTM(512, return\_sequences=True, input\_shape=(sequence\_length, vocab\_length)))
* model.add(Dropout(0.2))
* model.add(LSTM(512, return\_sequences=False))
* model.add(Dropout(0.2))
* model.add(Dense(vocab\_length))
* model.add(Activation('softmax'))
* model.compile(loss='categorical\_crossentropy', optimizer=’rmsprop’, metrics = ['accuracy'])
* checkpoint = ModelCheckpoint(filepath, monitor=’loss’, verbose=0, save\_best\_only)=True, mode=’min’)
* callbacks\_list = [checkpoint]
* history = model.fit(input\_notes, output\_notes, batch\_size=128, epochs=50, validation\_split=0.2, callbacks=callbacks\_list)
* PARAMETERS: 3,692,756
* TRAIN\_TIME: 1,149s
* loss: 1.7573 - acc: 0.4550 - val\_loss: 6.4089 - val\_acc: 0.0190



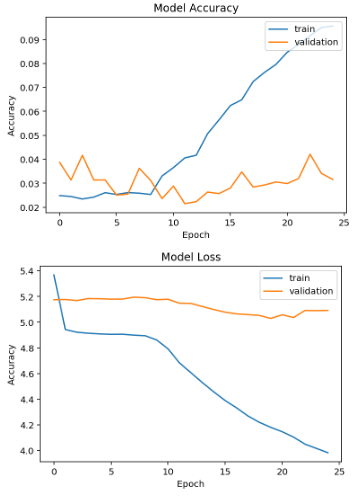
THIRD TRAINING: 30 songs

* model model = Sequential()
* model.add(LSTM(512, return\_sequences=True, input\_shape=(sequence\_length, vocab\_length)))
* model.add(Dropout(0.2))
* model.add(LSTM(512, return\_sequences=False))
* model.add(Dropout(0.2))
* model.add(Dense(vocab\_length))
* model.add(Activation('softmax'))
* model.compile(loss='categorical\_crossentropy', optimizer=’rmsprop’, metrics = ['accuracy'])
* checkpoint = ModelCheckpoint(filepath, monitor=’loss’, verbose=0, save\_best\_only)=True, mode=’min’)
* callbacks\_list = [checkpoint]
* history = model.fit(input\_notes, output\_notes, batch\_size=1024, epochs=25, validation\_split=0.2, callbacks=callbacks\_list)
* PARAMETERS: 4,491,788
* TRAIN\_TIME: 182s
* loss: 3.5774 - acc: 0.1478 - val\_loss: 5.2775 - val\_acc: 0.0398



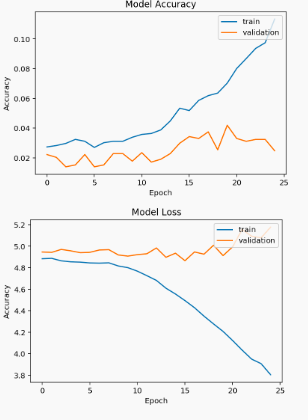
FOURTH TRAINING: 30 songs // used the adam optimizer, results not that great

* model model = Sequential()
* model.add(LSTM(512, return\_sequences=True, input\_shape=(sequence\_length, vocab\_length)))
* model.add(Dropout(0.2))
* model.add(LSTM(512, return\_sequences=False))
* model.add(Dropout(0.2))
* model.add(Dense(vocab\_length))
* model.add(Activation('softmax'))
* model.compile(loss='categorical\_crossentropy', optimizer=’adam’, metrics = ['accuracy'])
* checkpoint = ModelCheckpoint(filepath, monitor=’loss’, verbose=0, save\_best\_only)=True, mode=’min’)
* callbacks\_list = [checkpoint]
* history = model.fit(input\_notes, output\_notes, batch\_size=1024, epochs=25, validation\_split=0.2, callbacks=callbacks\_list)
* PARAMETERS: 4,491,788
* TRAIN\_TIME: 180s
* loss: 3.9828 - acc: 0.0956 - val\_loss: 5.0917 - val\_acc: 0.0316



FIFTH TRAINING: 24 songs // This was after re-cleaning the dataset

* model model = Sequential()
* model.add(LSTM(1024, return\_sequences=True, input\_shape=(sequence\_length, vocab\_length)))
* model.add(Dropout(0.2))
* model.add(LSTM(1024, return\_sequences=False))
* model.add(Dropout(0.2))
* model.add(Dense(vocab\_length))
* model.add(Activation('softmax'))
* model.compile(loss='categorical\_crossentropy', optimizer=’rmsprop’, metrics = ['accuracy'])
* checkpoint = ModelCheckpoint(filepath, monitor=’loss’, verbose=0, save\_best\_only)=True, mode=’min’)
* callbacks\_list = [checkpoint]
* history = model.fit(input\_notes, output\_notes, batch\_size=128, epochs=25, validation\_split=0.2, callbacks=callbacks\_list)
* PARAMETERS: 4,087,150
* TRAIN\_TIME: 127s
* loss: 3.0036 - acc: 0.1133 - val\_loss: 5.1772 - val\_acc: 0.0248



### Effort to evaluate your results

The model was evaluated with an 80/20 split of training and validation data. There were over 10 executions of the model yet best results of the first ten were logged for comparisons.

In my opinion the best output in regards to sound was the sixth execution on 25 epochs:

6- train\_output\_SequenceLength=25, Epochs=25, Optimizer=rmsprop, no\_dropout\_layers, songs=24 LSTM=512, Batch=128

testing\_acc: 0.4169 , val\_acc: 0.0299

### Effort to benchmark your method / results

Compared to advanced models like *Ji-Sung Kim's* *DeepJazz and Google’s Transformer model, the* improved *Transformer-XL* the performance of our model wasn’t as profound. The model was able to produce decent sounding results, yet as musicians we desire greater compositions. Due to the time constraints we were unable to implement some of the complex algorithms and architectures used in Deep Jazz and Transformer-XL models. Being able to comprehend the methods and architectures employed by the previous models would require many more hours of writing, debugging, testing the code, along with twice as much time training the model. Hearing the generated results from the two advanced models have enlightened us to understand that we still have room to progress.

### Documentation efforts

This report and the python notebook are documented well.

### Effort to document the training time

|  |  |  |  |
| --- | --- | --- | --- |
| **MODEL TRAINING** | **TOTAL PARAMS** | **TRAIN TIME: EPOCHS/\SECONDS** | **TRAIN TIME (MINUTES)** |
| FIRST: 6 FILES | 3,692,756 | 50/\1,149 | 19.5 |
| SECOND: 30 FILES | 4,468,739 | 25/\176 | 2.93 |
| FOURTH: 30 FILES | 4,491,788 | 25/\180 | 3 |
| FIFTH: 24 FILES | 4,087,150 | 25/\127 | 2.12 |

### Effort to study learning curves

SEE NUMBER 4.

### Effort to prepare a “reproducible” Python Notebook (ipynb) file

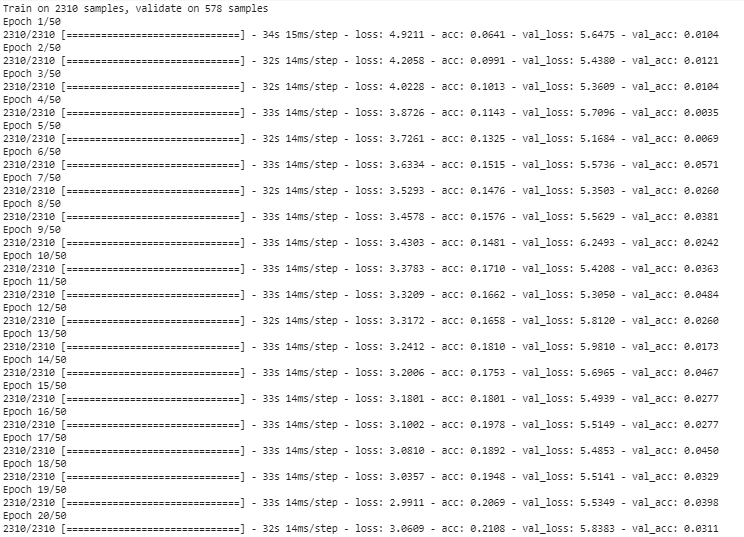
All files for this project, including a reproducible notebook are located in GitHub:

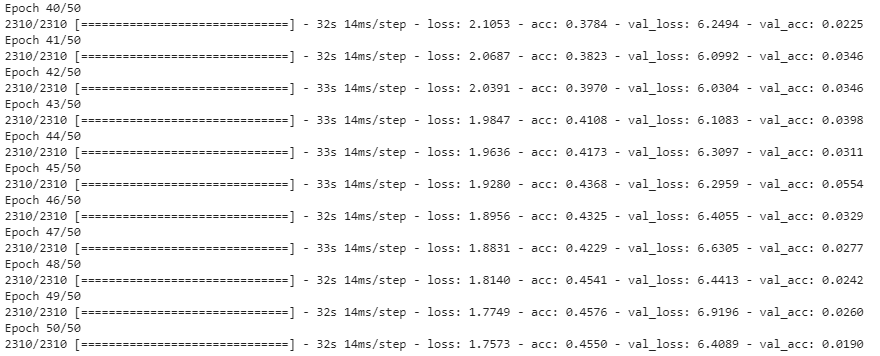
[https://github.com/code-front/JazzBot](https://github.com/code-front/BluesBot)

A direct link to the notebook is here:

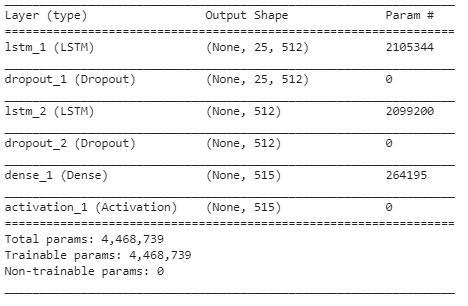
[https://github.com/code-front/JazzBot/blob/master/DeepLearningModel.ipynb](https://github.com/code-front/BluesBot/blob/master/DeepLearningModel.ipynb)

FIRST TRAINING:

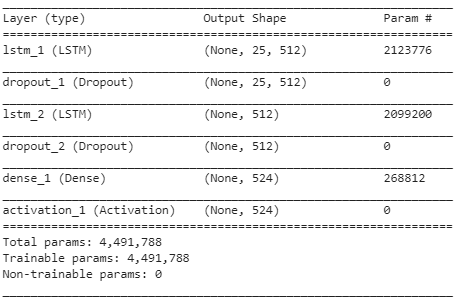


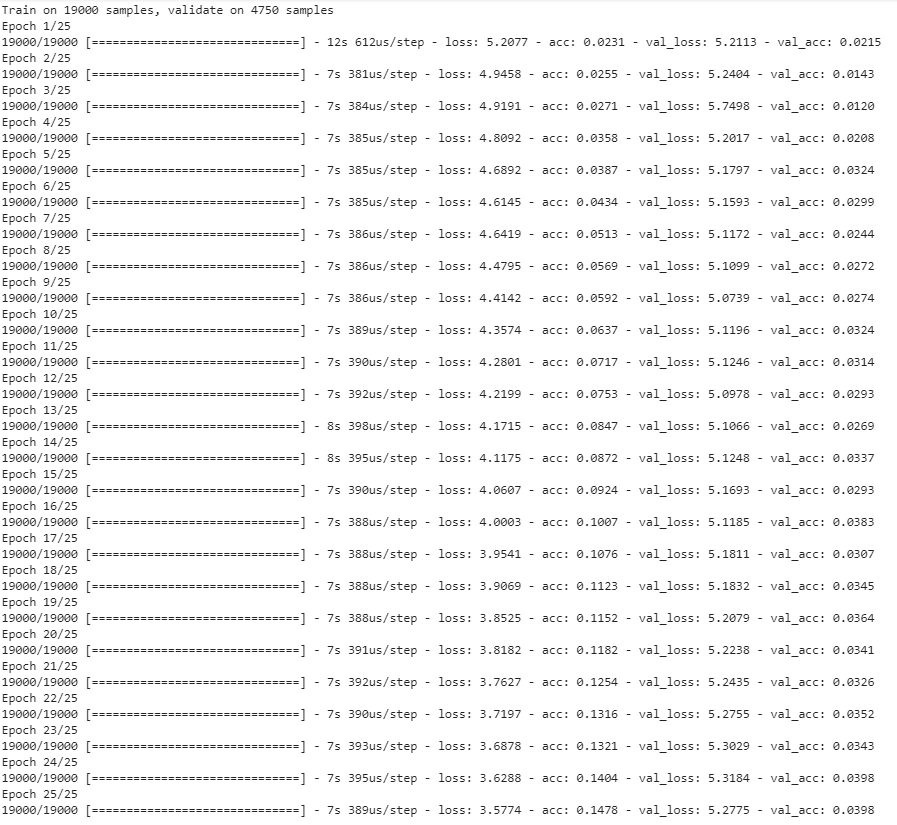


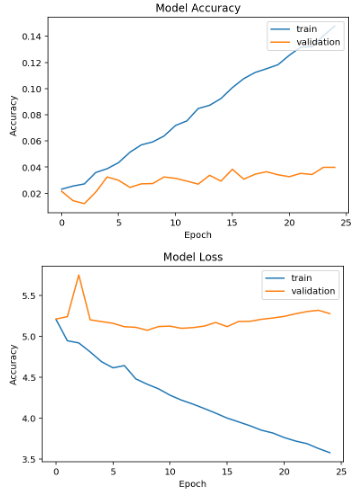
SECOND TRAINING:



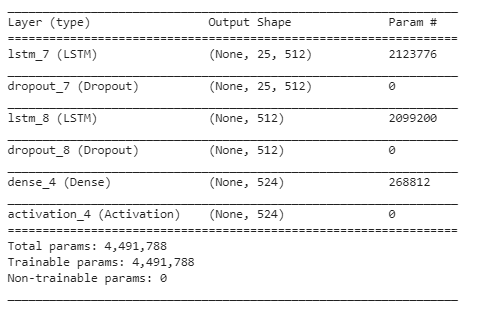
THIRD TRAINING:

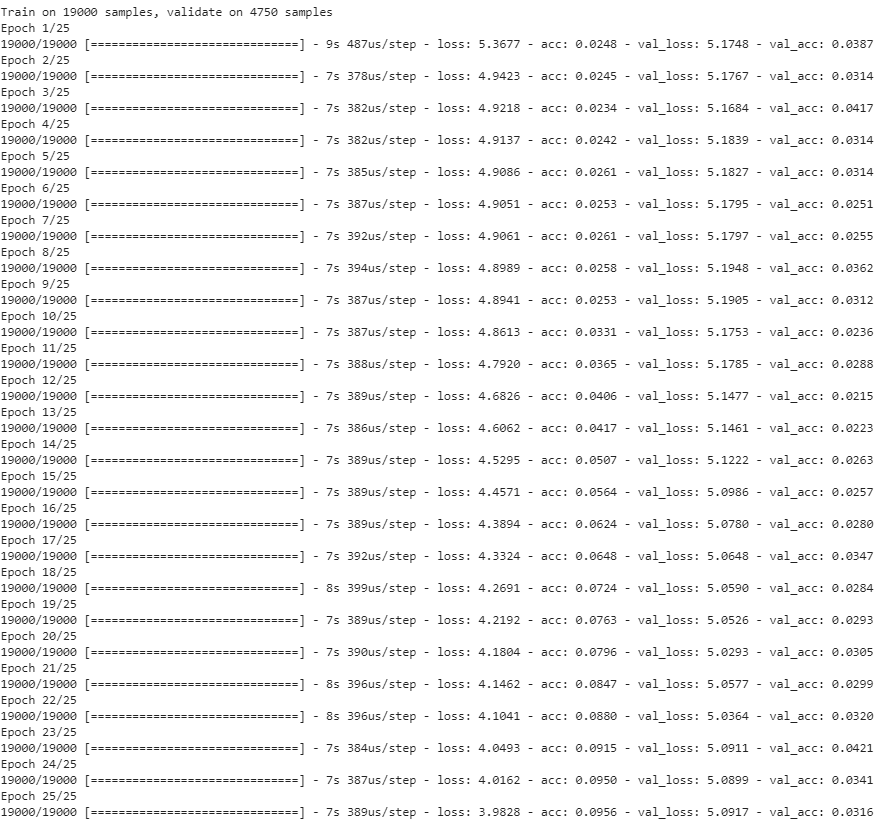


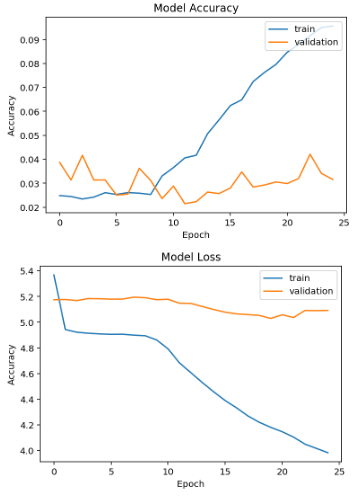




Fourth Training:

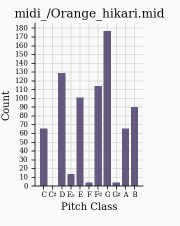


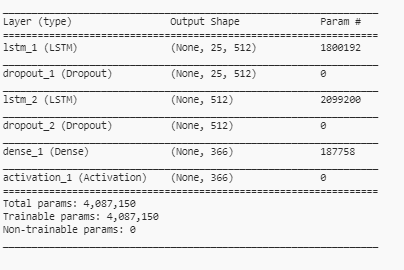


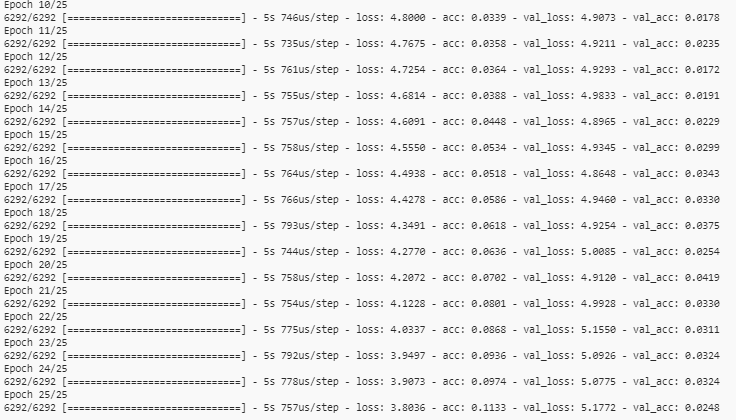


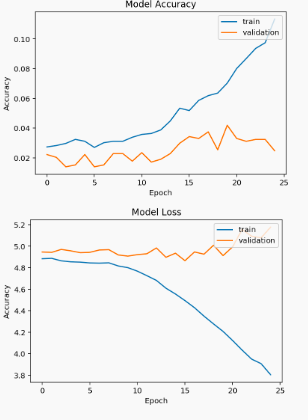
.

FIFTH TRAINING:









SIXTH TRAINING:

