**Objective**

The objective of using Tensorflow is to create a Recurrent Neural Network that is trained on kern data, and the objective of the Genetic Algorithm is to ‘randomly’ create new music that will be evaluated by the Neural Network. A Recurrent Neural Network was chosen because of the sequential structure of music. RNNs have layers that allow the network to have memory as it goes through each input sequence. Currently, a pair of Long Short Term Memory cells are included in the RNN to allow it to remember certain attributes as it is fed input. A Genetic Algorithm was employed for the actual creation of new music because of its ability to continually evolve the individuals of the given population. This ability to tweak created music based on the evaluation of the neural network would allow musical pieces to become better fit to the musical pieces that the neural network was trained on.

**Process**

The first step of the process is to prepare the data to be fed into the Recurrent Neural Network. As this research is still proof of concept, only basic features from the Kern files would be extracted to train the neural network. There are a pair of simple bash scripts, interval\_script and duration\_script, that extract only the intervals and durations for the first spine of each Kern file for a directory of related files. Next, all the stripped files, organized in meaningful directories, will be combined into a data structure that can be used directly to train the neural network.

The file, input\_preparation.py, does this work in a number of steps. First, it reads the stripped files containing the intervals and durations. As it does this, it creates an array with 3 attributes for each note: a rest is considered a note in this case. Intervals that are positive are modified by the equation 2n-2 and negative intervals are modified by the equation 2(-n) -1. Intervals are the first attribute of a note, and they are adjusted in this manner so that all unique intervals are represented as a positive number, making normalizing the dataset later on easier. A rest is notated as 0 for this attribute. (rest = 0, unison = 1, +step = 2, -step=3, +third=4, -third=5) Each duration is then separated to fill the other 2 attributes in the array. The first attribute for a duration is simply the duration without any dotted length added to it. (4. = 4, 4.. = 4) The second attribute then tells if the note is dotted or not. (not dotted = 0, dotted = 1, double dotted = 2)

After doing the above process for all the notes in a file, we append a class identifier for the given song so that the training neural network has a target for the song. The product of converting one song to this format results in something like the following: [[[0,4,0], [2,2,0], [4, 1, 0]], 0]. This full process is applied to every song, leaving an array containing all the songs in the format specified just above.

The dataset is almost in a format ready to be fed into the neural network, with the last thing needed is to normalize all the values to be between 0 and 1. After normalizing, we start training the network for between 20-40 iterations over the entire dataset. Roughly 20% of the dataset is set aside to validate the accuracy after each iteration. The weights for the model are the saved for later in the genetic algorithm, only being saved if they gave a better accuracy than the last saved weights. Multiple training attempts are made until the saved weights are satisfactory, with the current target being around 80% accuracy on the validation data.

After training, the next step is to actually attempt to make new music that would fit with a given class used to train the Neural Network. Each song created is deemed an ‘Individual’ in which its ‘Genes’ are the notes of the song. A genetic algorithm needs some sort of fitness function to decide how ‘fit’ each Individual is in the population. In this case, the previously trained neural network is used to predict the probability that an individual fits in each of the classes. The genetic algorithm is aiming to make songs that fool the neural network into thinking they belong in a given class. The genetic algorithm is currently designed around a 100-member population. After each generation, the top 10% automatically make it into the next generation. The rest of the members for the next generation are created by mating parents from the top 50% of individuals. This process involves selecting genes from both parents, with a chance of genes ‘mutating’ in the process. Mutations happen by adjusting one of the attributes of each note (either the interval, duration, or dotted). The mutations are currently designed to not be extreme. An interval mutation is more likely to adjust the interval by one, and a duration mutation is more likely to double or half the duration to keep a similar time feel as before. New generations are created until the fitness of the top individual surpasses a set threshold. Also, the top 10 Individuals at every 10-generation mark are saved for future reference.

**Future Work**

The current model for the RNN used is very simple, and an expanded network would be better at distinguishing the unique long-term characteristics that denote each genre/class. Extracting chords and all the spines for each file would also help distinguish characteristics found in each class. The genetic algorithm can be improved by adjusting the population size, individual carryover, and the chance and types of gene mutation.