1. **Introduction**

This research spawns from previous research attempts to create new musical scores from mass amounts of scores in the digital \*\*kern format. While previously created music can be said to sound subjectively better than attempts before it, it has been hard to reliably and objectively tell how good the new music is. The first objective of this research is to create a classifier that would score created pieces on how realistic they were. This itself would be very beneficial to any music generation system, and for this research, this classifier would also be directly used in the second objective, which is designing a system that more organically creates music. Previous attempts involved rule based systems with the music being created in one attempt; this research suggests implementing a process where better music will be created over generations. This is where the classifier will be used in the generative process; it will function as a discriminator that will be able to tell if a piece of music needs to be adjusted for it to be more realistic.

* 1. **Background**

To understand the process of this research, a background of the ideas, methods, and terminology employed will be needed. As is with all research, some paths chosen led to dead ends, but these attempts are not without valuable insight to the rest of the research process. One dead end attempt at a classifier involves Pink Noise, and its failings will be described in this section.

* + 1. **Terminology**

In the following research, there are terms and definitions that will be important to understand and not be confused with each other.  A note will be defined as a single musical unit containing the basic values of a pitch and a duration. Pitch is defined as the sound produced by a note. An interval for a note is derived from the distance between the last pitch and the current pitch. Duration is defined as the length of a note in time.  A rest is defined as a note that has no pitch but has a duration. A chord is defined as a grouping of notes played at the same time.

* + 1. **Pink Noise**

The phenomenon of Pink Noise was first explored as a potential classifier of musical scores in this research. The value of Pink Noise comes from an understanding of noise in general. In the surrounding world, noise can be usually be classified as white, pink, or brown. White noise can be thought of as the basis of random noise, with an example being the static sound that comes from a tv. Brown noise tends on the opposite side of white noise, where this type of noise can be too predictable and unnatural in sound. Pink noise falls in the middle of the two, giving this noise some predictably while maintaining some randomness to keep our interest; the idea is that pleasing sounds fall under the pink noise umbrella. We can analyze any noise in nature, or man-made, by using methods such as the Fast Fourier Transform (FFT). Using FFT, an input can be converted from the time domain to that of frequency. In this form, you can graph the curve produced, and a slope calculated. The slope of the curve can be used to classify the noise as falling under one of the three types of noise. The value of pink noise relating to music is that different songs can have varying levels of ‘pinkness,’ with the aim to show that the slopes for all the songs of a given genre fall in a set range. Classification would then be possible if the range of slopes for a genre did not overlap with those of other genres.

While promising results can be obtained showing different genres have slopes in a range unique to that genre, classifying songs this way disregards the importance of order in the musical scores. Scrambling the order of notes in a song would give the same slope for the song. This is an issue because one of the features that defines quality music is the order in which notes are played, not just the notes simply being present. Therefore, this potential classifier would fail in a number of ways. Firstly, a random grouping of notes could be viewed as fitting under a genre when it in fact does not. Secondly, a valid change to make a song better fit under a genre might be to swap notes around, but the classifier would not see a difference between the original and swapped version. The failing of using Pink Noise as a classifier leads this research on to a different classifier.

* + 1. **Neural Networks**

Neural Networks use multiple layers of processing to understand the data being fed through them. The first layer is the input layer, which is followed by any number of hidden layers that transform the input into something that can be used by the output layer. Neural Networks have a variety of uses which include classifying information and predicting outcomes. Different types of neural networks can be formed based on the structure of each layer and how the layers connect together. Traditional feed forward networks can perform well on input such as images, but they tend to perform poorly on sequential data. Since the order in which musical notes are played matters, a more specialized type of Neural Network would need to be employed for this use case. For this research, a Recurrent Neural Network (RRN) would be formed to allow the network to make sense of the sequential data that would be fed through it. Specifically, this RRN would leverage Long Short Term Memory (LSTM) cells in its layers. LSTM cells work by reading input, remembering certain aspects and forgetting others, and often output to multiple cells and/or layers. Moving forward with this research, a RRN will be used for classification.

* + 1. **Genetic Algorithms**

A Genetic Algorithm is a method used for solving optimization problems , and it is based on natural selection, the process behind biological evolution. This algorithm reflects the outcome of natural selection where the fittest individuals of a population are selected to produce offspring for the next generation. Genetic algorithms are composed of five steps: population initialization, fitness calculation, selection, crossover, and mutation.

The process of this algorithm begins with a set of individuals called the initial population. Each individual can be thought of as a potential solution to the problem to be solved or optimized. Ideally, the initial population should have great genetic diversity so that individuals have unique qualities many generations down the line.

The fitness function is crucial in a genetic algorithm. This function is used to calculate how fit an individual is. The score given to an individual determines the chances of that individual being selected for reproduction of the next generation.

In the selection step, the idea is that the fittest individuals will be used as parents to pass their genes on to the next generation. Individuals with higher fitness scores will have a higher probability of being selected for reproduction than those with lower scores. In this step, we will select as many groupings of two parents as needed, with individuals able to be parents more than once.

Crossover is the step where we decide which genes from each parent will be selected to create a new individual. Normally, crossover is decided by randomly selecting a point in the gene sequence of the parents from which a new individual will be created by taking the genes before the selected point from one parent and after this point from the other parent. Multi-point crossover can also be used in which many points are selected to divide up the gene sequence, making it possible to only select certain gene sequences from a given parent to pass on to the new individual.

Mutation is the final step in the process, and its importance is in making sure genetic diversity is maintained. After the new individuals are created from the crossover step, some individuals will undergo a mutation of their genes. This discourages convergence of the population, where new individuals of a population are too similar to those of the last generation. Mutation introduces new genes into the population, therefore allowing new genes that normally wouldn’t form during crossover.

This process of population initialization, fitness calculation, crossover, and mutation repeats until a specified number of generations have been made or an optimal fitness has been reached by an individual.

For this research, a genetic algorithm will allow musical pieces to improve over time while at the same discouraging undesirable musical traits. The ability to evolve musical pieces will be very important because correct musical characteristics may be in a piece, but they may not be in the right order or other parts of the piece might need to be slightly mutated. It is not as if musical masterpieces of the past were written in one go, it is more probable that they were constantly adjusted until the right sound and feel was acquired.

**1.1.5 Dataset**

The dataset for this research current consists of roughly 5000 songs in the \*\* kern format, with the dataset divided into two roughly equal genres of music, German folk and Chinese folk. The dataset of available \*\*kern files is much greater than the 5000 used, but the smaller selection was used as the research so far using a Recurrent Neural Network and a Genetic Algorithm is still proof of concept.

**2. Process**

The overall process is divided up into a number of steps. The first, and possibly most time consuming is data preparation. In this step, the information from the \*\*kern files must be extracted and rearranged into a format that can be understood by the neural network. The second step is to design and train the RNN on the prepared data. The final step is to create a genetic algorithm that generates songs that uses the RNN as a fitness function.

**2.1 Data Preparation**

The first step of the process is to prepare the data. As this research is still proof of concept, only basic features from the \*\*kern files would be extracted to train the neural network. \*\*Kern files are comprised of multiple spines that denote different musical voices, and each spine contains all the notes and chords for that voice. Only the first spine of songs are currently being used, and if a chord is in the spine, only the first note of that chord is used. As the files are stripped, we will store the interval and duration for each note separately.

Once the \*\*kern files are stripped, the information defining each note will need to be transformed into a format that can be used to train the neural network. Our new format for a note will have 3 attributes: interval, duration, dotted. The formatted interval is made by modifying positive intervals by the equation 2n-2 and negative intervals are modified by the equation 2(-n) -1. Intervals are adjusted in this manner so that all 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) The other two attributes in the formatted note are made by separating out the duration from the original note. The new duration attribute is simply the original 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 neural network knows the class while training. 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.

**2.2 Training the Recurrent Neural Network**

The neural network is a recurrent neural network because it has at least one layer of Long Short Term Memory cells; the network for this research is based on two LSTM layers with a few other layers used for the output.

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 best weights for the model are the saved for later in the genetic algorithm. Multiple training attempts are made until the saved weights are satisfactory, with the current target with this neural network being around 75% accuracy on the validation data.

**2.3 Creating the Genetic Algorithm**

To deploy a genetic algorithm, we need to denote what an ‘Individual’ will be and what its ‘Genes’ will be. For our case, it is as easy as deeming a song as 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 possible classes. The genetic algorithm’s goal is to evolve songs that fool the neural network into thinking they belong in a given class. Currently, a 100-member population is used. After each generation, the top 10% of individuals automatically make it into the next generation; this makes sure that the best genes stay in tact, with no worries of change caused by crossover or mutation. 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.

1. **Results**

The previous sections laid out how the system will work, and now it is time analyze how it has performed. Creating and training the RRN was quite successful. Even though the number of training examples is quite small in terms of most neural networks, the model was able to attain accuracy of around 75%. This is successful for a first attempt, and it shows that patterns are being found to classify songs.

The genetic algorithm as used in this research is also very promising. There is always the case that employing a genetic algorithm will not make better individuals because of its random nature, but that is not the case here. By starting with songs that have had notes randomly generated, we are able to create songs that were more fit with the classes used by the neural network. Generations that have all members with relatively low fitness scores are able to evolve into better fitting songs if given enough generations. This is promising in the fact that this organic creation of songs is actually able to increase the objective fitness, or quality, of songs based on the feedback of the neural network.

**3.1 Insights and Limitations**

Although the results of the recurrent neural network and genetic algorithm are quite promising, there are limitations in how the system is currently implemented. Regarding the dataset itself, there are a number of avenues that can be followed to increase the value and power of it. Using only the first spine of \*\*kern files is limiting as all the parts of a song contain more information and can give a better feel for the song than any part on its only. Combining this with the fact that only the first note of any chord is used leads to a one-dimensional representation of the song at hand. Therefore, expanding the input to include chords and all spines would add much needed dimensionality back to input used to train the neural network.

Another current shortcoming is the design of the RRN itself. The RRN is not badly designed, but it is very simplistic in its structure. Only a basic RRN was used for this research, and a more complex design with more intricate and fine-tuned layers would allow for a neural network to better understand longer range sequences. The current simplistic nature of the RRN might be causing incorrect training on the input data as it potentially oversimplifies correlations and patterns found in a given class of music. This oversimplifying could then cause issues as the neural network is used as a fitness function in the genetic algorithm; the RRN might be placing too much value on short and random patterns in generated songs. This could lead to poorly written music scoring a high fitness, even though it does not carry the long-term correlations and patterns that humans identify as necessary for a given class of music to sound pleasing.

Aspects of the genetic algorithm could also be improved. Varying the population size could be a start, as population size directly impacts the number of genes available. Worthwhile changes could be made in how carryover and mutation occur. Spending time adjusting how mutation affects genes could lead to potentially more beneficial changes in an individual/song.

Overall, this research has shown the power of employing a recurrent neural network and a genetic algorithm to create musical scores. The system in its current state has given useful feedback, but this research so far is only scratching the surface. Advancing and tweaking these systems has the chance to lead to far more interesting and valuable outcomes.