This experiment is to build a neural network that can identify the genre of a song. We're going to use what's called the GTZAN Genre Collection. It has 1000 different songs over 10 different genres. There are 100 songs per genre and each song is about 30 seconds long. Just to give you an idea of the kind of songs that are represented here, We're going to use the librosa library to extract features from the songs. We're going to use what's called Mel-Frequency Cepstral Coefficients or MFCC for short. MFCC values mimic human hearing and they're commonly used in speech recognition applications as well as music genre detection. These MFCC values will be fed directly into the neural net. We developed a helper function to display the MFCC values. First we load the song that's provided and then we extract the MFCC values from it. Then we use the specshow, which is spectrogram show from the librosa library. Here's the kick drum. We can see that at low frequency the bass is very obvious and at the rest it's kind of a wash. There's not much other frequencies represented whereas whistling it's pretty clear that there's higher frequencies represented. The darker the color or closer to red means more power in that frequency range at that time. So you can even see the kind of change in frequency with the whistles. Now here's the disco and another disco song. You can sort of see beats here, even though it's 30 seconds long so it's a little bit hard to see the individual beats. Compare with classical where there's not so much beats as a continuous kind of bassline like from a cello for example, and another classical. Here is hip-hop, and another hip-hop. It looks kind of similar to disco but if we required that we could tell the difference with our own eyes, we wouldn't really need a neural net because it'd probably be a relatively simple problem. We have another auxilary function here that again just loads the MFCC values but this time we're preparing it for the neural net. So we load the MFCC values for the song but because these values are between maybe negative 250 to positive 150, that's no good for a neural net. We don't want to feed in these large and small values. We want to feed in values near negative 1 & 1 or 0 to 1. So we're going to figure out what the max is, an absolute value for each song and then divide all the values by that max. Also, the song is a slightly different length, so we want to pick just 25000 MFCC values.We have to be super certain that what we feed into the neural net is always the same size because there are only so many input neurons and we can't change that once we've built the network. Next we have a function called generate \_features\_and\_labels that will go through all the different genres and go through all the songs in the dataset and produce those MFCC values and the class names. We're going to prepare a list for all the features and all the labels. Go through each genre of the 10. For each genre we're going to look at the files in that folder. This is how the dataset is organized. I'm going to say that we're processing that folder will be a hundred songs each and for each file we're going to extract the features and put those features in the list. The name of the genre for that song in a list also.So at the end, all features will have 1000 entries and all labels will have 1000 entries. In the case of all features, each of those 1000 entries will have 25000 entries. That'll be a 1000 x 25000 matrix. For all labels at the moment, it's a 1000 long list, and inside are words like this. Now, that's going to be a problem because a neural net is not going to predict a word or even letters. We want to give it a one-hot encoding, which means that each word here, that's going to be represented as ten binary numbers. In the case of the blues, it's going to be 1 and then nine zeros. In the case of classical, it's going to be zero followed by one, followed by nine zeros and so forth. First we have to figure all the unique names and get those back as integers. Then we have to do to\_categorical,which turns those integers into one-hot encoding. So what comes back is 1000 x 10 dimensions. 1000 because there are 1000 songs and each of those has ten binary numbers to represent the one-hot encoding. I'm going to return all the features stacked together into a single matrix, as well as the one-hot matrix. So we call that upper function and save the features and the labels. Just to be sure we're going to print the shape of the features and the labels. So it is 1000 by 25000 for the features and 1000 by 10 for the labels. Now we will split all the dataset into a train and test split. So we're going to decide at the 80% mark to do a split. Before that we want to shuffle and before we shuffle we need to put the labels with the features so they don't shuffle in different orders. We're going to call this alldata. Do the shuffle,split and then we have a train and a test.

Next we'll build the neural net. We're going to have a sequential neural network. The first layer will be a dense of 100 neurons. Now,just on the first layer it matters that you give the input dimensions or the input shape and that's going to be 25000 in our case. So that says how many input values are coming per example. Those 25000 are going to connect to the 100 in the first layer. The first layer will do its weighted sum of its inputs and its weights and its bias term and then we're going to run the relu activation function. relu, if you recall, anything less than 0 will turn out to be a 0. Anything above 0 will just be the value itself. These 100 will then connect to 10 more and that will be the output layer. It's 10 because we have one-hot encoding and specifically we have 10 binary numbers in that encoding. The activation here, our softmax, says 'take the output of the 10 and normalize them so that they add up to 1'.So that way they end up being probabilities and whichever one of the 10 is the highest scoring, the highest probability, we take that to be the prediction and that will directly correspond to whichever position that highest number is in. Next we compile the model, choose an optimizer like Adam, but there are alternatives. Next we can print the model.summary, which tells us details about the layers. We see that here.The output shape of the first 100 neuron layer is definitely 100 values because there are 100 neurons, and the output of the dense second layer is 10 because there are 10 neurons. Next we run fit. It takes the training input and training labels, takes the number of epochs that we want. We want 10 so that's 10 repeats over the trained input,takes a batch size which says how many in our case songs to go through before updating the weights and a validation\_split of 0.2 says 'take 20% of that trained input, split it out don't actually train on that and use that to evaluate how well it's doing after every epoch'. It never actually trains on the validation split but the validation split lets us look at the progress as it goes. Finally, because we did separate the training and test ahead of time, we're going to do an evaluation on the test, the test data, and print the loss and accuracy of that.