COMP 4106 – Final Project

Transcribing Music with a Convolutional Neural Network

Darren Holden

Problem Domain

The transcription of music is an important part of the music industry. Music transcription refers to the process by which natural music is recorded in a written notation. While a difficult process, it is useful as it allows for music to be shared, practiced, and saved for later. Transcription is also used to rewrite music for a different type of instrument. These impacts mean that the process of transcribing is a great asset to music as a whole, especially with the internet allowing transcribed music to be easily shared with many people.

Problem Motivation

Manual music transcription is a time-consuming and difficult process, so automating it is beneficial as it saves time and helps reduce human error. However, traditional automated music transcription often fails to match human performance [1]. AI techniques could be a solution to improving the accuracy of automated music transcription, as investigated by Sigtia et al. [1], and Bereket and Shi [2]. Even though automatically generated scores should be checked for errors, an accurate automated transcription process could cut the time required to transcribe music by a considerable amount. This project will use a convolutional neural network to generate a MIDI file from an MP3 of interest. MIDI files are widely used, and there are many tools available for converting them into scores or other formats.

AI Techniques

A convolutional neural network (CNN) was implemented in order to transcribe music. CNNs are well suited for image classification, which might seem at odds with the problem at first glance. However, converting audio files to a spectrogram yields an image representation of the audio. Using this spectrogram, a CNN can then be used to identify which notes are being played. For example, Figure 1 shows a partial spectrogram of Mozart's Sonata for Piano No. 11 in A Major.

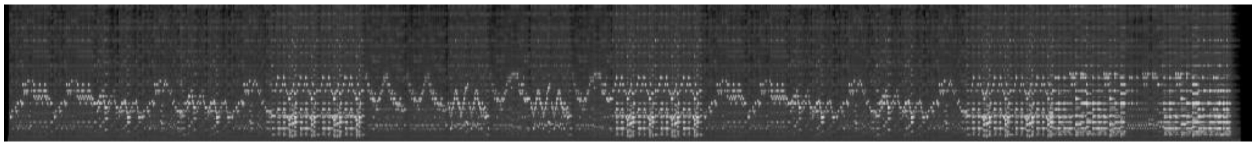

Figure 1: Partial Spectrogram of Mozart's Sonata for Piano No. 11 in A Major [Generated Using https://convert.ing-now.com/audio-spectrogram-creator/]

As can be seen in Figure 1, there are relatively clear indications of when different keys of the piano are being pressed. By stepping through the spectrogram in time slices, a CNN can be used to label each time slice with the notes that were played [2]. Once each time slice is labelled, the results of each time slice can be interpreted and combined to produce a score. When consideration each time slice, the surrounding area is also considered as the surrounding area can provide information on which notes are being played.

The data set for this problem would consist of the spectrograms and their corresponding scores, each divided into time slices. Dividing each song in the data set into time sliced representations will greatly increase the size of the data set, as each song will yield many time sliced values.

Using a CNN for images such as the spectrogram has some advantages over other types of images. In more traditional images, the orientation, aspect ratio, and size of the image are important and must be considered. For spectrograms, these are not concerns as there is a single “correct” orientation, and the notes can only occupy a given range, dictating the size of the image.

A recurrent neural network (RNN) system was also considered. RNNs are useful for speech recognition, which is similar in practice to music note identification, as explore by Sigtia et al. [1]. It was decided that a CNN would be used for this project as the spectrogram approach should be sufficient for the purposes of this project.

Mechanics of a Convolutional Neural Network

As the name suggests, convolutional neural networks are based around the concept of a convolution, which is an operation that can be performed on two matrices, resulting in a matrix. The first of these is the input matrix *I*, and the second is the filter, *F*. If matrix *I* has dimensions I_h by I_w by I_c and matrix *F* has dimensions F_h by F_w by F_c, then the dimensions of the output matrix *O* are determined as shown in Equation 1 [3], where the convolution is using a padding of *p* and a stride of *s*. The concepts of padding and stride will be explained later. It should be noted that the input and filter matrices need to have the same number of channels (I_c and F_c respectively), and that the output only has a single channel. For images, the number of channels is typically the number of colours used in the image. For example, a picture with standard red, green, and blue colours would have three channels, and a grayscale image would have one channel.

$$(O_h,\ O_w,\ O_c\ ) = \left(\frac{I_h+2p-F_h}{s}+1, \frac{I_w+2p-F_w}{s}+1, 1\right) \tag{1}$$

In order to calculate each element in the output matrix, Equation 2 is used [4]. This is illustrated in Figure 2.

$$O_{i,j} = \sum_{a=0}^{F_h-1}\sum_{b=0}^{F_w-1}\sum_{c=1}^{F_c} I_{i+a,j+b,c} \cdot F_{a,b,c} \tag{2}$$

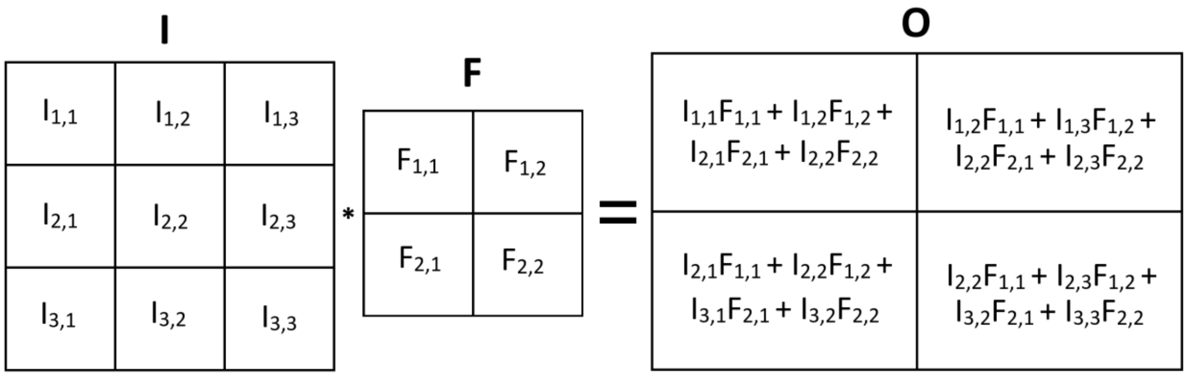
I
F
O
*
=
I1,1 I1,2 I1,3 I2,1 I2,2 I2,3 I3,1 I3,2 I3,3
F1,1 F1,2 F2,1 F2,2
I1,1F1,1 + I1,2F1,2 + I2,1F2,1 + I2,2F2,2
I1,2F1,1 + I1,3F1,2 + I2,2F2,1 + I2,3F2,2
I2,1F1,1 + I2,2F1,2 + I3,1F2,1 + I3,2F2,2
I2,2F1,1 + I2,3F1,2 + I3,2F2,1 + I3,3F2,2

Figure 2: A Convolution of a Single Channel Input with a Filter

As previously mentioned, there are two additional parts to a convolution: the padding and the stride. Padding is the process of adding zeroes around the outside of the input matrix. For example, if one layer of padding were added to matrix *I* in Figure 2, it would become a five by five matrix where the outermost columns and rows consist solely of zeroes. Padding the matrix in this way has several benefits. The first is that this can allow the output matrix to have the same dimensions as the input matrix, which can help to preserve information. The second is that this ensures that the outermost columns and rows of the unpadded input matrix to be used in just as many convolutions as the inner values.

Stride refers to how much the filter moves between each round of the convolution. In Figure 2, a stride of one was used. A bigger stride could be used, provided that the input matrix is big enough to support it. For example, if matrix *I* had dimensions of four by four, a stride of two could be used instead of a stride of one. In this case, the filter would move two spaces at a time. Stride helps to reduce the size of the output image and can reduce the overall amount of computations.

In a CNN, convolutions are performed in a convolution layer. This layer contains a set of filters and convolves each of them with the input that it receives. Each element in the resultant matrix is also appended with a bias, which is unique for each filter. It produces an output matrix for each of its filters, meaning the number of channels in the output is equal to the number of filters. The output is then passed to the next layer. The output matrix is also subject to an activation function. In this project, a rectified linear unit (ReLU) function was used. This sets all negative values in the output matrix to zero.

Convolution layers are normally followed with pooling layers [3]. Pooling layers are meant to reduce the dimensionality of the input, which reduces the number of parameters needed at deeper layers. The pooling layer looks at a block of the input and generates a single output value for the block. This output value is usually the maximum or average value in the block. For this project, a maximum function was used. This is pictured in Figure 3.

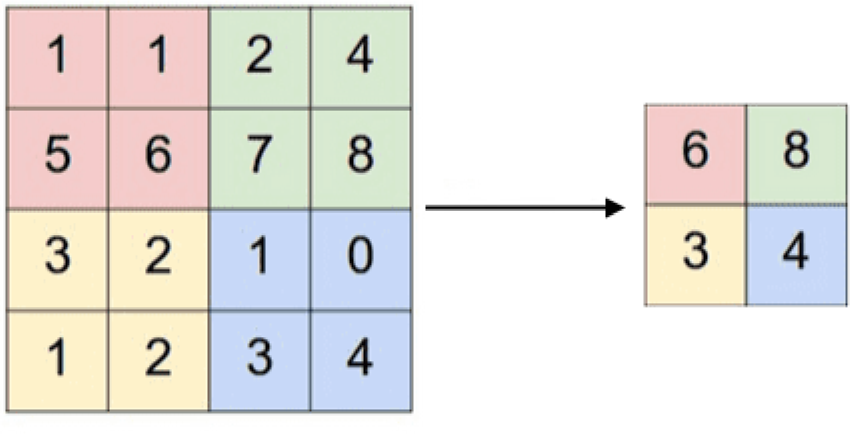

Figure 3: Demonstration of the Max Pooling Function [4, Modified]

The convolution and pooling layer pairs can be repeated many times, but after the final pooling layer there is a series of fully connected layers, the last of which generates the final output. These fully connected layers are identical to the layers of a more traditional neural network. Each fully connected layer consists of a number of nodes. Each of these nodes receives all of the output values from the previous layer. Each value is multiplied by a weight, and the products are then summed. Simply put, this is a dot product between the input array, and the transposed array of weights. The result is then passed through an activation function which generates the output for that node. In this project, the activation is a Sigmoid function. The output of the layer is an array of the outputs from each node. A simple fully connected layer is pictured in Figure 4.

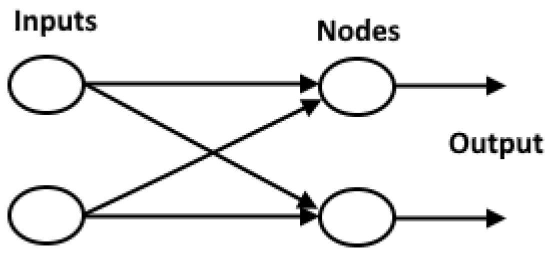

Figure 4: A Simple Fully Connected Layer with Two Nodes

Backward Propagation

In order to train the neural network, errors in the result must be propagated back through the layers. The initial errors are determined simply by subtracting the neural network's output from the expected output. These errors are then passed into the last fully connected layer. The error on each of the weights in the layer is determined by multiplying the matrix of all the weights with the product of the error array and the output array. The updated weights are determined by summing each of the original weights with the product of the corresponding error and the learning rate. The errors which are propagated back are determined by taking the product of the transposed, original weight matrix, and the errors. This is repeated for each of the fully connected layers.

In the pooling layers, there are no parameters to be adjusted by the error. However, the error must be distributed so that it is associated with the input which got through the pooling. Only the parameters which were used to calculate the value which was the maximum should be updated. To achieve this, the input errors are distributed to a matrix according to the pooling function [5]. This is shown in Figure 5, which shows how errors would propagate back for the situation depicted in Figure 3.

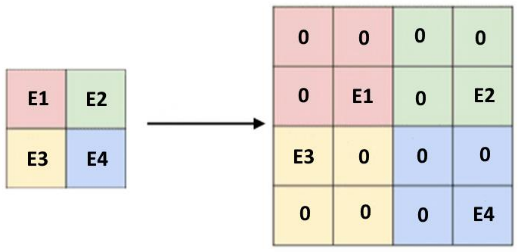

Figure 5: Back Propagation for a Max Pooling Layer

For the convolution layer, the simplest portion of the backpropagation is updating the bias. The error on the bias is simply the mean of the errors on the output generated by the corresponding filter. To update the bias, the bias's error is multiplied by the learning rate and then added to the bias's value. The error on the filters is calculated by performing a convolution on the original inputs, and the errors. The filters are then updated by multiplying the filter errors by the learning rate and adding them to the filter values. The error to be propagated is determined by performing a convolution on the original inputs and the filter errors.

System Design

The system designed in this project has a fairly simple architecture. It consists of a single convolution layer, followed by a pooling layer, followed by two fully connected layers. This is pictured in Figure 6. This figure also shows the shape of the data as it moves through the network.

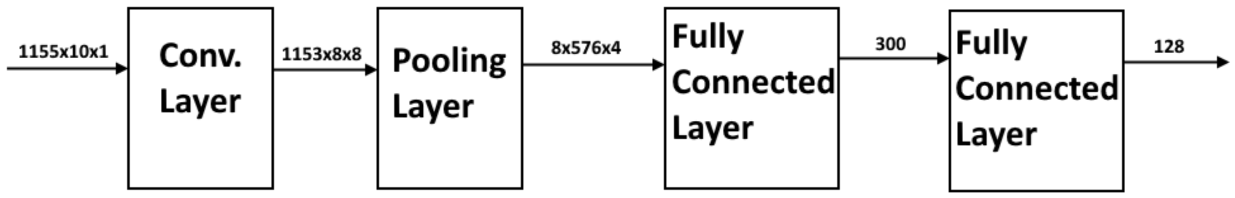

Figure 6: Neural Network Structure Showing the Change in Data Shape as it is Processed

The input to the system is 1155 by 10 pixel slices of the spectrogram generated from an MP3 file. This size encompasses full frequency range and captures the area around the time of interest. The spectrogram is in grayscale, so it only has a single channel.

The convolution layer uses eight 3 by 3 filters for its convolutions. It uses a padding of zero, and a stride of 1. These choices stem mostly from the small width of the input. Having a larger filter size would decrease the size of the output more, meaning that there is less information for the deeper layers. The stride is set to one for a similar reason. The use of eight filters gives an output with eight channels.

The pooling layer looks a two by two sub-matrices of each channel of the input, which essentially halves the dimensions of each channel. Two by two is smallest square sub-matrix which makes sense for pooling on. Pooling on a larger sub-matrix would further reduce the dimensions of the output, which, given the low width of the initial input, was found to be less than ideal.

There are then two fully connected layers which work the values down to the 128, which is the number of possible notes. The final output is an array of values indicating which notes the neural network thinks is being played.

Results

In order to test the system, a data set of 30.5 thousand samples was collected. Each of these samples consisted of a slice of the spectrogram, and the notes which were being played during it. The data set was split so that 90% of the samples were used for training, and 10% were used to test the accuracy of the neural network. The data set was randomly shuffled before the split was done. The accuracy after each round of training can be seen in Table 1.

Table 1: Accuracy of the Neural Network After Several Rounds of Training

Training Rounds Completed	**Accuracy**				
	0 Mistakes	**1 or Less Mistakes**	**2 or Less Mistakes**	**3 or Less Mistakes**	**4 or Less Mistakes**
0	0%	0%	0%	0%	0%
1	12.049%	67.372%	87.764%	96.824%	100%
2	12.049%	67.372%	87.764%	96.824%	100%
3	12.049%	67.372%	87.764%	96.824%	100%

As seen in the table, further rounds of training beyond the first had no effect on accuracy. There is a substantial accuracy increase after a single round of training. The accuracy was calculated with five different perspectives, each considering the number of mistakes, either being a false positive or a false negative. In this case, a false positive occurs when the neural network predicts a note that is not correct, and a false negative is when the neural network does not predict a note that it should have.

At first glance, the results seem rather lacklustre, and to some extent they are. Having to dig through all of the results to weed out the mistakes would be a fairly challenging task. However, having two mistakes means the result is still 98.5% correct. An example of the complete values can be seen in Figure 7. Clearly, the output is mostly wrong, however it did get the first note correct.

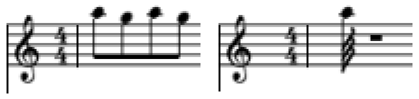

Figure 7: Comparison of the Original (Left) to the Output from the Neural Network (Right)

Testing showed that adding an additional convolutional layer between the existing convolutional and pooling layers did not improve performance. All of the accuracies for the modified architecture matched those shown in Table 1, within a tolerance of 1%.

Enhancements

This project could be enhanced in several ways. The first is that a larger data set could be used, which could introduce more varied situations into the system.

Further experimentation with different architectures could also be performed. The architecture used for this project is probably not the ideal architecture for this problem. The most intuitive way to find a better architecture would be to test them and compare the results, however this rather time-consuming process.

A final enhancement would be to try different activation functions. For example, average pooling may be better than maximum pooling. The activation functions in the convolutional and fully connected layers could also be investigated.

References

[1] S. Sigtia, E. Benetos, and S. Dixon, “An End-to-End Neural Network for Polyphonic Piano Music Transcription,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 24, no. 5, pp. 927–939, 2016.

[2] M. Bereket and K. Shi, “An AI Approach to Automatic Natural Music Transcription,” Stanford University, n/d. [Online]. Available: http://cs229.stanford.edu/proj2017/final-reports/5244388.pdf. [Accessed Apr. 15, 2020].

[3] P. Sharma, “A Comprehensive Tutorial on Convolutional Neural Networks (CNNs),” Analytics Vidhya, Dec. 27, 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/. [Accessed: Apr. 16, 2020].

[4] M. Rathi, “Convolutional Neural Networks,” Sep. 4, 2018. [Online]. Available: https://mukulrathi.com/demystifying-deep-learning/convolutional-neural-network-from-scratch/. [Accessed: Apr. 16, 2020].

[5] J. Kafunah, “Backpropagation In Convolutional Neural Networks,” DeepGrid, Sep. 5, 2016. [Online]. Available: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/. [Accessed: Apr. 16, 2020].

Appendix: Running the System

The system runs using Python 3.7. Other versions of Python 3 will probably suffice, but they have not been tested. On top of Python, several other libraries:

- NumPy;
- Matplotlib;
- Mido;
- SciPy; and
- Imageio.

Each of these libraries can be installed using the pip tool, which comes default with Python.

To process an MP3 file, simply place it in the "ToProcess" folder. Then run the RunAssessment.py script. It will load a saved instance of the convolutional neural network. This prevents a long wait time for training a new model. The neural network will then process each slice of a spectrogram, which is generated from the MP3 file. The script will then stitch the sequence of results together in order to form a MIDI file, which is a representation of when each note is being played.

Technical Details

File	Description
Main.py	Creates the data set, shuffles it, splits it into training and test sets, trains and tests the CNN, and then saves the CNN using pickle
DataSetGenerator.py	Creates pairs of data points between the spectrogram and the notes played.
MidiParser.py	Reads a Midi file and determines which notes are being played at which times.
SpectrogramGeneration.py	Generates a spectrogram from a specified MP3 file.
ConvolutionalNeuralNetwork.py	Contains the class for a CNN, as well as classes for the three types of layers. The CNN constructor creates a CNN with a hardcoded structure.
RunAssessment.py	Uses the CNN saved by the Main.py script in order to generate a Midi file for each MP3 located in the Music/ToProcess folder