AUTOMATIC MUSIC TRANSCRIPTION WITH CONVOLUTIONAL NEURAL NETWORKS USING INTUITIVE FILTER SHAPES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jonathan Sleep

October 2017

COMMITTEE MEMBERSHIP

TITLE: Automatic Music Transcription with Con-

                            volutional Neural Networks using Intuitive

                            Filter Shapes

AUTHOR: Jonathan Sleep

DATE SUBMITTED: October 2017

COMMITTEE CHAIR: John Clements, Ph.D.

                            Professor of Computer Science

COMMITTEE MEMBER: Dennis Sun, Ph.D.

                            Professor of Statistics

COMMITTEE MEMBER: Alexander Dekhtyar, Ph.D.

                            Professor of Computer Science

ABSTRACT

Automatic Music Transcription with Convolutional Neural Networks using Intuitive
Filter Shapes

Jonathan Sleep

This thesis explores the challenge of automatic music transcription with a combination of digital signal processing and machine learning methods. Automatic music transcription is important for musicians who can't do it themselves or find it tedious. We start with an existing model, designed by Sigtia, Benetos and Dixon, and develop it in a number of original ways. We find that by using convolutional neural networks with filter shapes more tailored for spectrogram data, we see better and faster transcription results when evaluating the new model on a dataset of classical piano music. We also find that employing better practices shows improved results. Finally, we open-source our test bed for pre-processing, training, and testing the models to assist in future research.

ACKNOWLEDGMENTS

I would first like to acknowledge and thank my advisor, Dr. John Clements, who always kept me grounded and gave me good feedback for pushing forward. He always let me know about any problems I might face on my approach, and I always regretted it when I didn't listen. Thanks again for all the help Dr. Clements!

I'd also like to thank my girlfriend, Annabelle, who gave me a ton of support to finish this thesis, especially at the bad times. Thank you Annabelle, I love you so much.

I finally would like to thank my parents, Richard and Kathy. Without their help and encouragement, I wouldn't have made it this far. Thank you for everything Mom and Dad, I love you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Music is a universal language that can be expressed in different forms. One form of music is the sound produced when performed on an instrument. Similar to natural language, another way music is represented is through transcription on paper (sheet music) or a digital medium (MIDI/MusicXML).

Having music represented in a transcribed format is important for a variety of reasons. One reason is that it can be a tremendous aid for musicians that have not memorized a composition. A composer can distribute instrument-specific sheet music to an entire orchestra ensemble so that they can perform it without having to memorize it. Another use is in musicology, where one might want to study a certain motif from a specific composer. In digital transcription formats like MIDI, a musician can directly edit and manipulate the transcription to their liking with a Digital Audio Workstations (DAW). The digital format can then be synthesized into audio using libraries of different sounds to apply on the note sequences. Just like natural language, transcription is important to music as well.

In the Classical era, composers like Mozart could listen to a musical performance just once and transcribe it[28]. While this may be easy for masters of music, the ability to transcribe must be learned. Until it is learned, novice musicians must rely on transcriptions from other sources.

Recent advances in machine learning and natural language processing (NLP) have brought about new applications for language classification and generation. Common to most smart phones now is a question answering system, that can listen and respond meaningfully to questions. In these systems, there are three main components: Speech

Recognition or Speech-To-Text (STT) to recognize what words were being said from audio input, Natural Language Processing (NLP) to infer meaning and produce a suitable answer, and Text-To-Speech (TTS) to return a response in a human-like way. This paper aims to perform an equivalent of Speech Recognition for music, commonly known as automatic music transcription (AMT).

The term "automatic music transcription" was first used by audio researchers James A. Moorer, Martin Piszczalski, and Bernard Galler in 1977. With their knowledge of digital audio engineering, these researchers believed that a computer could be programmed to analyze a digital recording of music such that the pitches of melody lines and chord patterns could be detected, along with the rhythmic accents of percussion instruments. The task of automatic music transcription concerns two separate activities: analyzing a musical piece, and printing out a score from that analysis[29].

We present an automatic music transcription system through the use of deep neural networks and digital signal processing techniques that try to exploit the nature of music data, the background of which are discussed in depth in Chapter 2. Automatic Music Transcription has been researched copiously and we will discuss the current state-of-the-art and its limitations in Chapter 3. We build off a previous model and make several design choices that we hypothesize should improve the ability to transcribe. In Chapter 4, we will discuss the design of the new model which will perform the AMT. In Chapter 5, we explain the observation and results from training and testing our model. Finally, we conclude our findings in Chapter 6.

Chapter 2

BACKGROUND

In this chapter, the relevant topics of Digital Signal Processing and Machine Learning will be discussed in detail.

2.1 Sound and Digital Signal Processing

2.1.1 Sound

Sound is what we experience when the ear reacts to a certain range of vibrations. The two main qualities of sound are pitch and volume.

Pitch is the quality of sound which makes some sounds seem "higher" or "lower" than others. It is determined by the number of vibrations produced during a given time period. The vibration rate of a sound is called its frequency. The higher the frequency, the higher the pitch. Frequency is often measured in units called Hertz (Hz). An instrument's perceived pitch is normally made up of the lowest frequency, called the fundamental frequency, and harmonics that are multiples of the fundamental.

Loudness is the amount, or level, of sound (the amplitude of the sound wave) that we hear. Changes in loudness in music are called dynamics. Sound level is often measured in decibels (dB).

Timbre is another important aspect of sound which allows us to distinguish between different sound sources producing sound at the same pitch and loudness. Each different musical instrument has a different timbre, and depending on how it is played, the same some instrument could produce different timbres as well.

2.1.2 Music and MIDI

Music is a form of sound. In western music (which is used exclusively in our evalua-
tion), there are 12 pitch classes each distanced a semitone or half step between each
other. A note is one of these pitch classes, in a particular octave (range of frequen-
cies), that occurs for a duration of time. A chord is defined as two or more notes
played at the same time. In this thesis, we perform music transcription for piano so
we deal with the piano roll notes, which are 88 keys, from note A0 to C8.

We make use of MIDI, or Musical Instrument Digital Interface. Specifically we
make use of reading from and writing to MIDI files, which contain sparse note on and
note off events for a particular pitch and velocity (intensity of how note is played).
The MIDI pitch range is from 0-127 but we only look at the range between MIDI
notes 21-108 which represent A0-C8.

2.1.3 Digital Signal Processing

Digital Signal Processing (DSP) is an engineering field that focuses on the computa-
tional methods for analyzing and altering digital signals. A digital signal (like one for
audio) comes from an analog continuous signal which gets sampled at a set sampling
rate. Sampling rates should be large enough so that our digitally sampled audio can
accurately represent the continuous audio. If we sample too little, much information
about the signal can be lost. Once we have a digital representation of the signal, we
can then perform operations on it.

2.1.3.1 Discrete Fourier Transform

One of the most important operations within the realm of digital signal processing is the Discrete Fourier Transform, given by equation:

$$X[k] = \frac{1}{N} \sum_{j=0}^{j=N-1} x[j] * e^{-j(\frac{2\pi}{N})nk} for \ k = 0 \ldots N-1$$

The purpose of the DFT is to take in N samples and generate N frequency bins that denote the frequency content of the entire collection of samples. See Figure 2.1 where a single sinusoidal function, denoting that it has only one frequency present, is fed into the DFT. From the resulting magnitude, we see a single peak denoting the frequency contained within the signal. This operation is vital for the task of automatic music transcription.

It is also worth mentioning that when the DFT is computed - it is standard to use the Fast Fourier Transform (FFT) algorithm. FFT sizes must be powers of 2. The algorithm runs in $O(n * log(n))$ time complexity rather than the $O(n^2)$ we see in the original discrete Fourier transform equation.

2.1.3.2 Short-Term Fourier Transform

While the DFT is powerful by itself, it only produces a a single frequency spectrum. If we were to perform DFT on a entire song, we would have some idea of what frequencies exist, but no idea when they occur. For transcription, it is necessary to produce a frequency spectrum often enough to see when when notes start and end.

In this case, we can use the Short-Term Fourier Transform, which basically computes DFT over a full signal, in small time segments. The parameters are described in Table 2.1. The main takeaway is that by using the STFT we can measure the frequency content over a signal and how it changes over time. STFT is a common
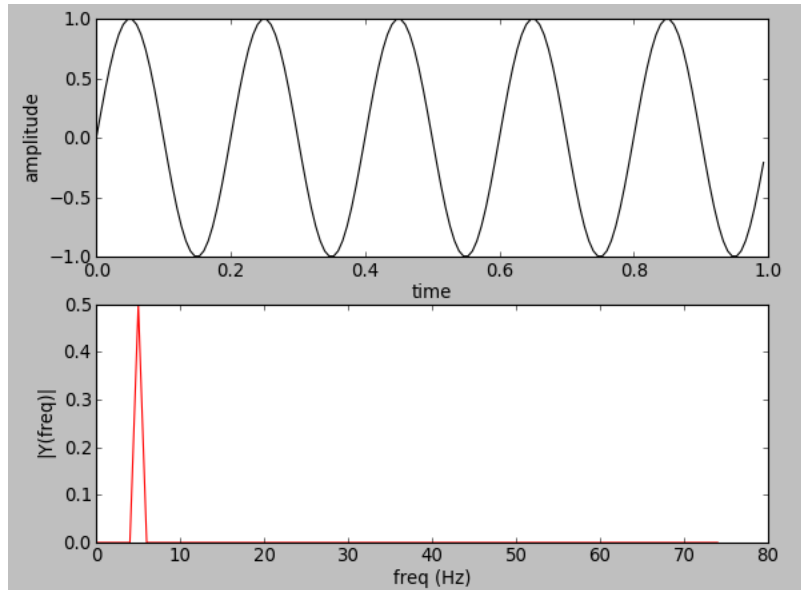
Figure 2.1: Example DFT of a Sinusoidal function with a frequency of 5 Hz

way to compute spectrograms of an audio signal, an example of which is shown in Figure 2.2. Having a time-frequency representation of audio is also vital to the AMT system.

2.1.3.3   Constant-Q Transform

Another way to represent frequency over time of an audio signal is to use the Constant-Q transform[4]. The transform is well-suited for handling musical data and this can be seen in Figure 2.3. Note how the distance between harmonics increase exponentially based on pitch in the STFT spectrogram, but remains constant in the CQT spectrogram. We chose not to display the STFT frequencies on a log scale because the frequency bins are not organized by the log scale. However, it is possible to easily display STFT on a log frequency scale.

The transform exhibits a reduction in frequency resolution with higher frequency bins. Because the bins are logarithmically spaced, fewer frequency bins are required to cover a given range effectively, and this proves useful where frequencies span several

Figure 2.2: Example of audio waveform with computed STFT of C Major Scale
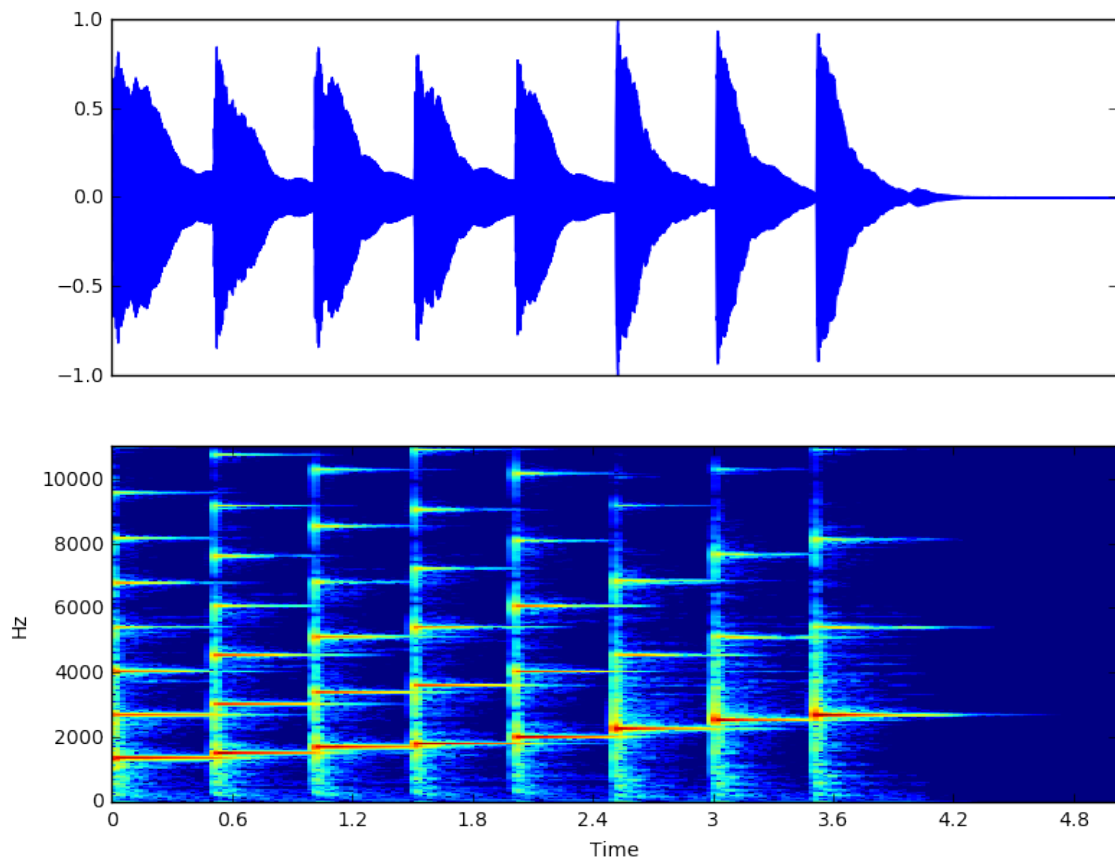
Table 2.1: STFT Parameter descriptions

Parameter	Purpose
FFT Size	Number of samples to use for FFT, must be a power of two
Hop Length	How many samples to 'hop' before perform the next FFT. This will usually be somewhere between a quarter to half of FFT size to get optimal overlap.
Window	The function to use on the slice of samples before computing the DFT. The windowing function zeros outs the ends of the slice so that adjacent transformations will be more continuous.
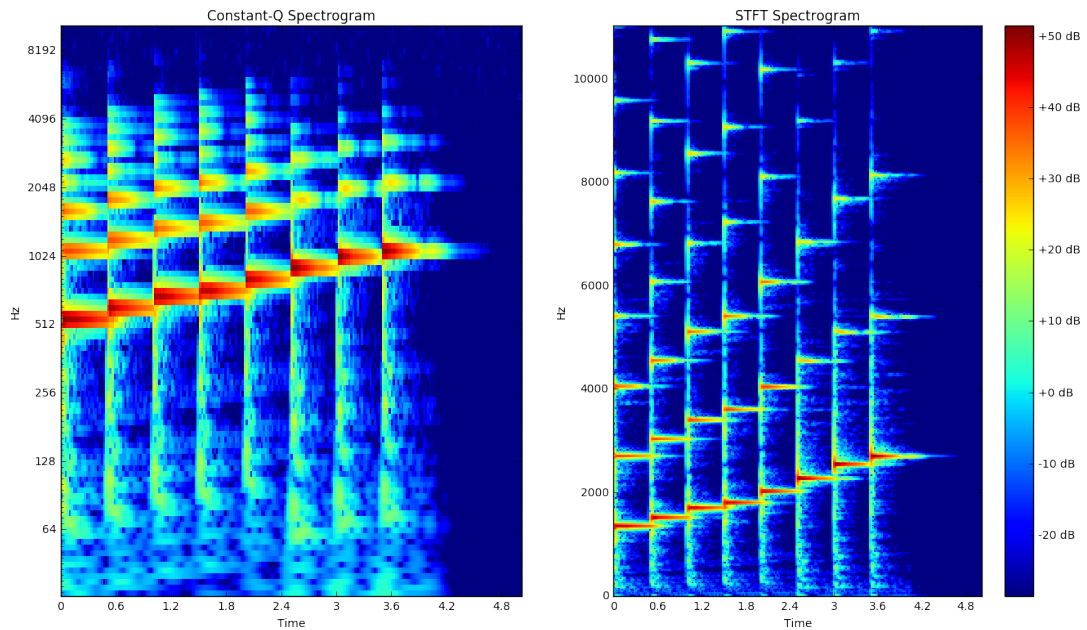
Figure 2.3: Comparison of CQT (left) and STFT spectrograms of C Major Scale
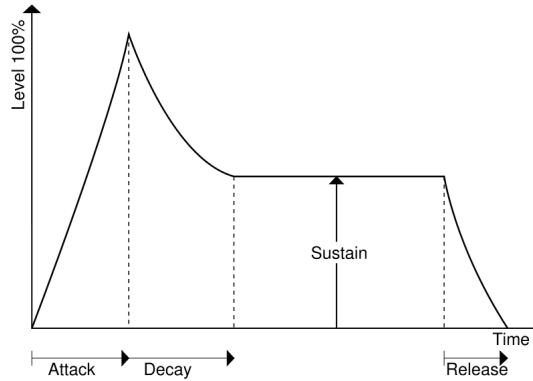
Figure 2.4: The Attack Decay Sustain Release Model

octaves. As the range of human hearing covers approximately ten octaves from 20 Hz to around 20 kHz, this reduction in output data is significant.

2.1.3.4 ADSR Model & Onsets

The Attack/Delay/Sustain/Release (ADSR) Model depicts how a musical event, such as a string being plucked, changes over time. When an acoustic musical instrument produces sound, the loudness and spectral content of the sound change over time in ways that vary from instrument to instrument. The "attack" and "decay" of a sound have a great effect on the instrument's sonic character[9]. Our system will need to learn these different stages of sounds in order to determine when a particular note is active.

Onsets and Offsets in Music Information Retrieval (MIR) are when music events begin and end, respectively. By using onset and offset detection, we can focus attention on sections of audio that when a music event is taking place.

A standard to detecting onsets uses the *SuperFlux*[2] onset detection method. This onset detection algorithm computes a spectral flux onset strength envelope with filters for maximum vibrato suppression. Once we have the strength envelope, we can pick peaks of maximum strength to be considered onsets, or at least onsets that

9

are in the process of transitioning from the attack to the decay stages in the ADSR model.

Estimating offsets or releases for monophonic instruments is fairly easy as we just wait for the energy produced from the last onset to subside until the next one starts. However, sometimes the energy is not completely released by the time the next onset starts, and that needs to be taken into account. Estimating offsets are different as for polyphonic instruments, such as piano, as some notes are ending while others are still sustained. One way to handle this is by tracking pitch from onsets and determining when specific frequencies subside[20].

However, our model will simultaneously perform onset / offset detection as well as pitch estimation by sweeping through all audio frames and determining if a note is active at the current center frame being analyzed.

2.1.3.5 Mono-pitch estimation

Monophonic pitch estimation is more or less a solved problem. Autocorrelation was the first employed methodology and is still used in many applications to determine the fundamental frequency, and thus pitch, of the instrument.

Autocorrelation is the correlation of a signal with a delayed copy of itself as a function of delay. Informally, it is the similarity between observations as a function of the time lag between them. The analysis of autocorrelation is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies.

The YIN algorithm is used for the estimation of the fundamental frequency (F0) of speech or musical sounds[7]. It is based on the well-known autocorrelation method with a number of modifications that combine to prevent errors. The algorithm has
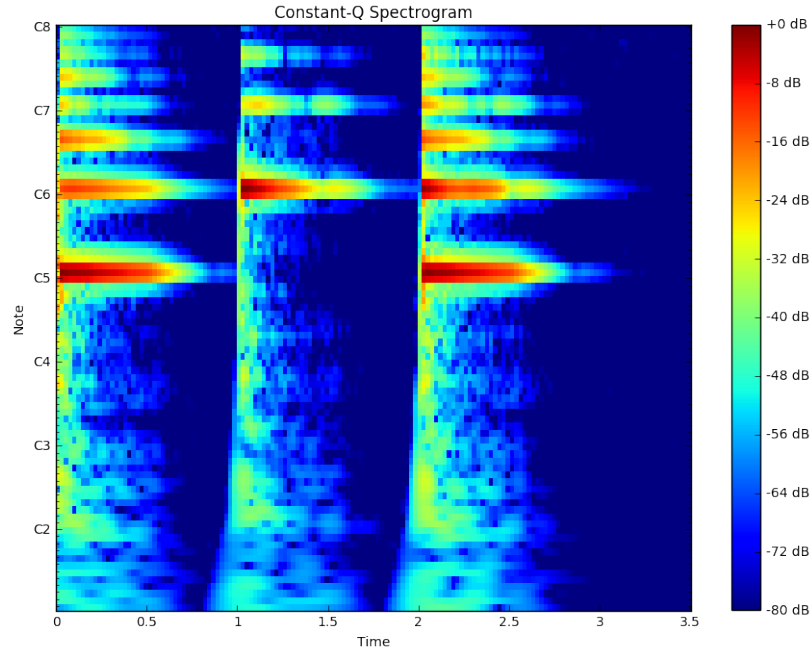
Figure 2.5: A C5 note, C6 note, and both being played

several desirable features. Error rates are about three times lower than the best competing methods, as evaluated over a database of speech recorded together with a laryngograph signal. There is no upper limit on the frequency search range, so the algorithm is suited for high-pitched voices and music. The algorithm is relatively simple and can be implemented efficiently and with low latency and involves few parameters that require adjustment. It is based on a signal model (periodic signal) that can be extended in several ways to handle various forms of aperiodicity that occur in particular applications. Finally, interesting parallels can be drawn with models of auditory processing.

2.1.3.6 Multi-pitch estimation

Multi-pitch estimation is the hardest component of automatic music transcription. One significant challenge is when notes of the same pitch class are played simultaneously. In these instances, the harmonics of the highest note reinforces the harmonics

of the lowest note, instead of creating new peaks in the spectrum to the point that one could hear a single note with a different timbre. When mixing several notes together, some of their partials can reinforce or cancel each other. See Figure 2.5 to see the differences between just a C4 note, just a C5 note and both C4 and C5 notes. Differentiating both notes being played from just the C4 note being played is a bit of a challenge as they both have the same active harmonics. Another challenge of multi-pitch detection is not knowing how many notes to find, unlike monophonic detection for determining a single note. For this reason, polyphonic pitch estimation can not be solved using autocorrelation based methods. If we are to use a time-frequency representation like STFT or CQT presented before, it is basically impossible to hand-write rules to handle all scenarios that need to be addressed for transcription. However, one way to discover these rules and have a system use them for AMT is through machine learning.

2.2 Machine Learning

The present potential for more accurate AMT results from the progress made in the fields of both digital signal processing and machine learning. Machine learning is a sub field of computer science that involves learning from data and using that trained system to make predictions on new data. The work of a machine learning practitoner mostly deals with framing a problem to use machine learning, preparing data, and training and testing a designed model on the data while tuning parameters for optimal results. Many successful products incorporate machine learning when explicitly programming rules is not contextually feasible.

This work will make use of a particular approach to machine learning, supervised learning. Supervised learning takes in pairs of correlating inputs and outputs that a system will learn to approximate. The system trains on these pairs until it converges
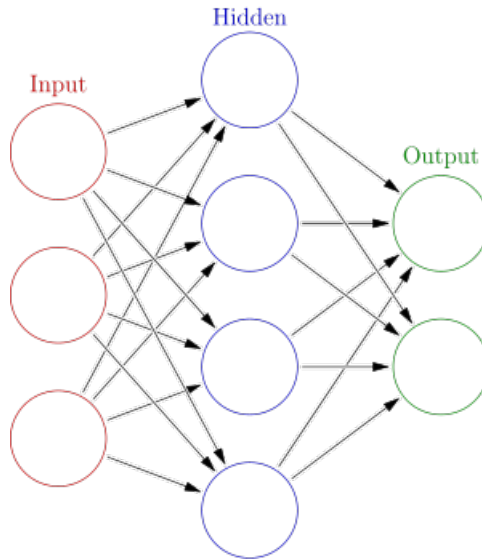
Figure 2.6: Example Neural Network Model

to an error rate that is satisfactory. We can then test the system on data it has not used before to determine how well the network can generalize new data.

2.2.1 Artificial Neural Networks

We will make use of a specific supervised learning approach, Artificial Neural Networks. Artificial Neural Networks is a model that loosely represents the brain as a network of neurons and the connections between them.

We first discuss plain vanilla or fully connected neural network models. Plain vanilla neural networks are composed of input, output, and a configuration of one or more hidden layers. Each layer in a simple neural network is fully connected with the last, meaning that every node of one layer shares a connection with each layer of the next. Each of these connections are weights that are learnable parameters, meaning that they change over time during the training process.

sigmoid(x) =

1/(1+e^-x)

tanh(x) =

2/(1+e^-2x) - 1

ReLU(x) =

max(0,x)
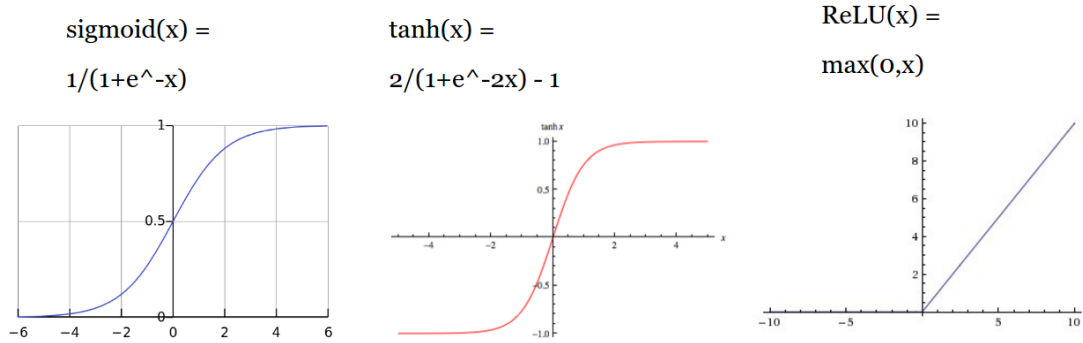
Figure 2.7: Popular Activation Functions

2.2.1.1 Forward Propagation

To go from input to output, neural networks go through the process of forward propagation. In forward propagation, we take in each input and multiply that input by the weight or connection that it shares with each node of the layer, also called hidden layer. Each node in the next layer becomes a weighted sum of all the nodes or inputs from the layer before it. This weighted sum then goes through an activation function, such as the sigmoid function. This process repeats until it hits the output layer for prediction.

2.2.1.2 Activation Functions

Activation functions are applied after performing multiplication between the inputs and weights of each fully connected layer. They squash the weighted sum of each value in the resulting vector into a defined range. Figure 2.7 shows some popular activation functions. The work we base our model off of uses the sigmoid and tanh, while in our model we use the ReLU activation function, for reasons we discuss in future sections.

2.2.1.3 Prediction

In the most simple classification problem, a neural network will try to determine from features if it is or isn't a certain class. The score outputted from the system would normally be in range from 0 to 1 and we can just round the prediction out to either a 0 or 1 choice. However, when trying to classify into multiple classes, it is common practice to use a softmax layer that takes each output probability, converts them to exponents, and follows by normalizing each probability so that they sum to one.

For automatic music transcription, we are trying to determine which notes are activated in a group of spectrogram frames, meaning that there can be more than one note being played at a time. In the context of neural networks, this is framed as a multi-label (because more than one note can be active at any given time). Here we cannot use the softmax layer because we are not just trying to choose the most probable single class. We instead try to create a single threshold to determine whether or not each note is active based on the score of the output neuron corresponding to that note. We can determine this threshold by having our system predict note probabilities over the training data inputs and comparing the predictions of the system to the ground truth or binary vectors. The threshold is a single number that makes the predictions match the ground truth as close as possible.

2.2.1.4 Backward Propagation

During the training process, we need to measure how close the predictions are to the ground truth or correct labels. We do this by using a differentiable loss function. This loss function informs us how wrong our output probabilities were (before prediction). By making this function differentiable we can use an optimization function such as stochastic gradient descent to update the weights of the neural network with correct values by using the derivative of the loss function which should help minimize the loss.

The weights connected to the output layer are corrected first and then the weights from the input to the hidden layer, which is in backward or reverse order from forward propagation.

2.2.2 Convolutional Neural Networks and Deep Learning

This work will specifically make use of Convolutional Neural Networks (CNN), which have shown good results in image classification tasks. CNN's are very similar to plain vanilla neural networks from the previous subsection: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with an activation function. The whole network still expresses a single differentiable score function: from the features on one end to class scores at the other. Just as for vanilla neural networks, they still have a loss function (e.g. Softmax) on the last (fully-connected) layer.

Plain vanilla neural networks do not scale well for structured data, such as 2D dimensional data like images or spectrograms. If a simple neural network were to take in a 7x352 spectrogram slice, this would result in taking 7x352 = 2464 input features. If we were to use 500 neurons in the next layer, this would result to 500*2646=1232000 weight connections, which is a wasteful amount of connectivity that will probably lead to the neural network overfitting. We need to exploit the structure of the data.

Convolutional Neural Networks take advantage of the fact that the input consists of images or audio spectrograms and they constrain the architecture in a more sensible way. In particular, unlike a simple neural network, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. For example, the input images in CIFAR-10, a popular image dataset, are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). The neurons in a convolutional layer will only be connected to a small region of the layer before it,
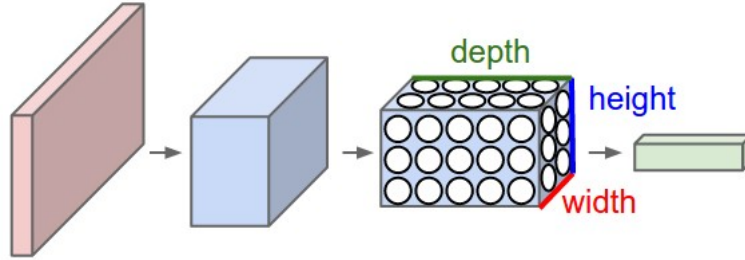
Figure 2.8: Example Convolutonal Neural Network Model

instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the CNN architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

There are 3 different types of layers or operations involved in a typical convolutional neural network model. The first is the convolutional layer. The convolutional layer takes a number of filters and convolves the filters around the input; each filter computing a dot product between their weights, and each small region they can be connected to in the input volume. Parameters involved in this operation are the filter shapes, the number of filters, the distance that the filters stride when convolving over the input, and the amount of zero padding on the inputs to keep the volume width and height constant. If we were to use the CIFAR-10 [32x32x3] RGB image with a convolutional layer with 12 filters of any size with zero padding to keep the height and width constant, the output volume would be [32x32x12]. In our case, spectrograms don't have color channels, so our input volume depth is only 1. After a convolution layer is applied, an activation function is performed, generally the ReLU activation function. An activation function does not change the size of the data. Another common layer is the pooling layer. A pooling layer will perform a downsampling operation along the spatial dimensions (width and height), by either taking the max or average of the values in the pooling regions. Performing max pooling of 2x2 on the last output

will result in an output volume [16x16x12]. A common CNN architecture generally follows the pattern of

$$INPUT->[[CONV->ACT]*N->POOL?]*M->[FC->ACT]*K->FC$$

where the * indicates repetition, and the POOL? indicates an optional pooling layer. Moreover, $N >= 0$ (and usually $N <= 3$), $M >= 0$, $K >= 0$ (and usually $K < 3$)[16].

2.2.3 Case studies

There are several popular architectures in the field of Convolutional Networks. The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990s. Of these, the best known is the LeNet architecture[19] that was used to read zip codes, digits, etc. The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton[17]. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). AlexNet had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single convolutional layer always immediately followed by a pooling layer).

2.2.4 Deep Learning

Deep learning is the term coined in the 2010's to signify the stacking of many layers in neural networks for the purpose of building a hierarchy of concepts or features. With plain vanilla neural networks, there was never a reason to use more than 2 layers, as any nonlinear function could be approximated with just 2 layers. However, as learned in image recognition tasks, building a hierarchy of concepts with more than 2 convolutional layers is helpful, as seen with AlexNet. Figure 2.9 shows an example
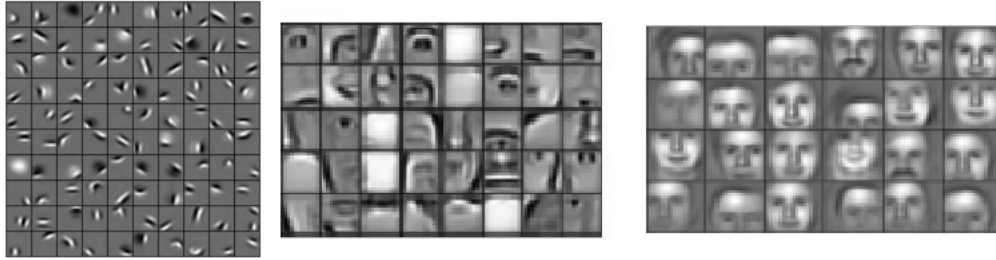
Figure 2.9: Hierarchy of Features learned with a deep face classifier

of features learned in different layers of a face recognition network. It learns low level features like lines and curves in the first few layers, then moves on to more abstract features like body parts, and eventually could have full faces. Deep Learning has only been made possible with the ability to utilize Graphics Processing Units, or GPUs, to greatly parallelize the computation needed for forward and back propagation in neural networks, as it can be framed as a sequence of matrix multiplication jobs. GPUs excel at parallelizing operations like matrix multiplications for much faster computation than CPUs.

2.2.4.1   Good Practices

There are several good practices that should be followed when designing and training neural network models.

Generally speaking, it is better to use the ReLU activation function over the sigmoid and tanh functions, as the ReLU is less likely to be saturated or stuck in the minimum/maximum value range of the function. The sigmoid is also not zero-centered which is generally a desirable feature for propagating values through network, especially if you have many layers, where saturated values can accumulate as it flows through the network. AlexNet found that using ReLUs allowed their network to converge faster by a factor of 6[17]. ReLU functions have also shown better results specifically for acoustic models [21].

A recently developed technique by Ioffe and Szegedy called Batch Normalization[15] alleviates many problems with properly initializing neural networks by explicitly forcing the activations throughout a network to take on a unit Gaussian distribution. When inputting data into a neural network, it is standard to keep features of the data scaled by computing the mean and standard deviation of each feature across the training dataset, and for each batch used for training and testing to subtract by this mean and divide by the variance to keep features scaled as best as possible. However, as data flows through the neural network and is multiplied by weight layers, the features start to become unscaled again. Inserting Batch Normalization layers between the weight layers and before the activation functions helps by keeping the features scaled throughout the network.

How you regularize a neural network can also be important for the network to generalize for all data instead of over-fitting to the training data. One method of regularization is using dropout[33], where based on a probability parameter, the operation will zero out some inputs of the intermediate data. This makes the network resilient to depending on one specific feature for classification as it has to learn to train from any combination of features that make it through the dropout layer. It is only actively dropping values in the training phase.

Unfortunately, neural networks, especially deep ones, are hard to train because it is difficult to determine ahead of time which hyper-parameters are best. One of the most important hyper-parameters for training a neural network is the learning rate. How it is initially set and scheduled to decay over training can also greatly help in the training process[18].

Chapter 3

RELATED WORKS

In this chapter, several approaches to automatic music transcription relevant to this
work are discussed.

Arguably, the first major AMT work was Smaragdis et al.[32]. This approach uses
Non-Negative Matrix Factorization (NMF). NMF works by factoring a nonnegative
matrix $\mathbf{X} \in \mathbb{R}^{\geq 0, MxN}$ (such as the magnitude spectrogram of audio) into two factor
matrices. One chooses how many components R to be used, such as 88 in the case
of piano roll transcription. The algorithm will then try to factor the matrix into two
matrices: $\mathbf{H} \in \mathbb{R}^{\geq 0, MxR}$ and $\mathbf{W} \in \mathbb{R}^{\geq 0, RxN}$. $\mathbf{H}$ represents the activations, or when
each component is active. $\mathbf{W}$ represents what each component is, for magnitude
spectrums this is just a frequency spectrum of what should only be a single note.
See Figure 3.1 for an example score with the factored matrices. Similar to supervised
learning approaches, a cost function is used to measure how much error the product
of the factored matrices resembles the target matrix. The cost function used in this
work is:

$$C = |X - WH|$$

While this is the main methodology employed in much software for automatic tran-
scription, it has its limitations. For example, it needs to know how many individual
notes are desired for the transcription in question - either by forcing to provide the
full range of individual isolated notes or from knowing which notes are present in
the audio ahead of time (both of which are not always available). Also, the authors
provide very little evaluation of their work, and fail to provide metrics or available
datasets to test against for future work.

The next work worth mentioning is Emiya et al.[10]. While we will not discuss

21
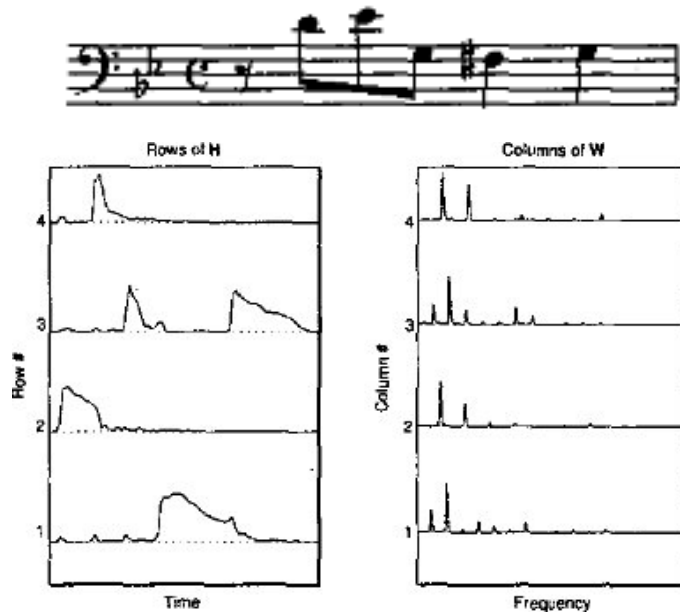
Figure 3.1: Example Score with NMF

the actual design of their transcription system (as it was out-performed by the next discussed work in the same year), we do note that the dataset they created has become the standard in evaluating any multi-pitch estimation system. In their work, Emiya et al. create the MIDI-Aligned Piano Sounds (MAPS) data set composed of around 10,000 piano sounds either recorded by using an upright Disklavier piano or generated by several virtual piano software products based on sampled sounds. The dataset consists of audio and corresponding annotations for isolated sounds, chords, and complete pieces of piano music. For our purposes, we use only the full musical pieces for training and testing neural network acoustic models. The dataset consists of 270 pieces of classical music and MIDI annotations. There are 9 categories of recordings corresponding to different piano types and recording conditions, with 30 recordings per category. 7 categories of audio are produced by software piano synthesizers, while 2 sets of recordings are obtained from a Yamaha Disklavier upright piano. These are recordings that are automatically performed, but recorded in a room from an acoustic piano (with its hammers hitting the strings when keys are pressed), simulating a more

realistic environment for actual transcription. This dataset consists of 210 synthesized recordings and 60 real recordings. We will use this dataset as the means of evaluation and comparison as have most other transcription systems since its release.

In the same year as the prior work, Vincent et al.[35] published a novel transcription system using a NMF-like method with an extension using probabilistic latent component analysis (PLCA). PLCA aims to fit a latent variable probabilistic model to normalised spectrograms. PLCA based models are easy to train with the expectation-maximisation (EM) algorithm and have been extended and applied extensively to AMT problems. In their paper, they model each basis spectrum as a weighted sum of narrow band spectra representing a few adjacent harmonic partials, thus enforcing harmonicity and spectral smoothness while adapting the spectral envelope to each instrument. They derive a NMF-like algorithm to estimate the model parameters and evaluate it on the MAPS database of piano recordings, considering several choices for the narrowband spectra. Their proposed algorithm performs similarly to supervised NMF using pre-trained piano spectra but improves pitch estimation performance by 6% to 10% compared to alternative unsupervised NMF algorithms. This model has remained the state-of-the-art for roughly five years.

Certainly the most relevant work to this thesis is Sigtia et al[30]. In this work, the researchers built the first AMT system using convolutional neural networks (among others) and outperforming the state-of-the-art approaches using NMF. Convolutional Neural Networks are a discriminative approach to AMT, which has been found to be a viable alternative to spectrogram factorization techniques. Discriminative approaches aim to directly classify features extracted from frames of audio to the output pitches. This approach uses complex classifiers that are trained using large amounts of training data to capture the variability in the inputs, instead of constructing an instrument specific model. We construct a baseline model based on the description of optimal parameters supplied from this paper (described in Table 3.1). This paper uses two

Table 3.1: Optimal Parameters & Architecture for ConvNet in Sigita et al.[30]

Parameter Name	Value
Window Size (# Spectrogram Frames)	7
# of ConvNet (Conv+tanh+Pooling) Layers	2
# of Fully Connected Layers	2
Window Shapes (1,2)	(5,25),(3,5)
Pooling Size	(1,3)
Convolutional Layer Filter #	50,50
Fully Connected Widths (1,2)	1000,200

types of models, an acoustic model for computing pitch probabilities for each frame of the magnitude spectrum, as well as a Music Language Model (MLM), to exploit the sequential nature of music for deciding whether a pitch should actually be transcribed, based on the previous notes transcribed. The paper also proposes a hash beam search method so that the hybrid architecture of the two models can be used with pruning and hashing, decreasing the time needed to transcribe significantly. This method outperforms Vincent et al.[35] in F1 Score, when evaluated per spectrogram frame. However, when evaluated on a per note basis, Vincent has a better F1 score.

While the paper introduced many novel ideas, we propose multiple changes: exploiting knowledge of the data and using filter shapes that are intuitively designed for audio data and employing good practices in the design and training of neural networks. Accordingly, we discuss our design of these changes in the next chapter.

Chapter 4

SYSTEM DESIGN

In this chapter, we discuss how we approach designing the proposed automatic music transcription model.

4.1 Preprocessing

Before even using a neural network model, data must be obtained and/or modified for it to train and evaluate.

We create a preprocessing script that takes in audio files, as well as matching MIDI or text files for constructing input/output pairs for the neural network. An audio file be stored as an MP3 or a WAV file (other types as well). These files hold a 1-dimensional array of number that represent sound pressure over time. We use the Librosa library[23] to arbitrarily handle any type of audio file, downsample it to a sampling rate of 22.05 kHz (Half of CD-quality), as well as convert the file to mono-channel.

With the audio samples loaded into memory, we then need to compute a time-frequency representation of the audio. We choose CQT over STFT because the CQT has logarithmically spaced bins. This is advantageous as the harmonic pattern stays constant for each note, or is pitch invariant, which makes training the neural network much easier. To compute the Constant-Q transform, we again use the Librosa library. Depending on the model we train, we will use different parameters for the CQT, depending on how many bins per octave we choose to compute. Sigtia et al.[30] uses 36 bins. We choose 48 bins per octave, which gives us better frequency resolution. 48 is also a multiple of 12 which allows us to downsample the volume twice into

the appropriate size where each bin corresponds to a MIDI note. We make up for the time resolution by using multiple frames of spectrogram for analysis. We also attempt to compute more total bins than the baseline frequency bins by calculating the maximum amount of bins as possible based on our sampling frequency. We were able to compute a total of 400 bins, which corresponds to range between roughly 30Hz -11KHz.

For pre-processing the data used for training, we also parse through accompanying MIDI files using the pretty_midi library[26]. This library includes a convenient function to take a MIDI file, which is mostly just filled with disparate note-on and note-off events including pitch, velocity, and time, into a piano roll or time-note representation. The function includes a useful parameter 'times', that permit sampling MIDI files at specific times. This is vital to matching each window of audio frames to a sample of the MIDI file. Once we have this representation, we clip the values from 0 to 1 (which excludes the velocity information) to get binary vectors that accurately depict which notes are active at which times. To get these times, we need only to know the shape of the audio spectrogram, specifically how many frames are present, and then compute the times at which they occur using our chosen sampling rate and hop size.

Once we have these representations, our network can then directly ingest them. We save these into data files that can be efficiently loaded using NumPy's memory map functionality. We chose these types of files and file loading over others as we are accessing the memory in sequential slices. We save roughly 30 songs worth (from each category of the MAPS dataset) of spectrogram and MIDI data into each data file to be used at training and testing time.

4.2   Proposed Models

In this section, we will discuss the proposed new models. Before doing so, it it is appropriate to review the Sigtia model in order to separate their work from our own.

Sigtia et al. were the first to apply convolutonal neural networks to automatic music transcription. Their model takes in a window of spectrogram frames as input to estimate what pitches are present in the center frame. It has been shown that using a window of frames instead of an individual frame as context helps with analysis[3]. The model will output a score for whether a note is active at the given center frame. By inputting in a spectrogram calculated from the full audio of music, we can perform automatic transcription by calculating the scores, post processing the scores to binary vectors to find which notes are active in a frame, and then converting the piano roll or note-time representation into a MIDI file that is used as transcription. We adopt their model and make various changes, the most novel of which uses intuitive filter shapes that try to better exploit the audio data.

Like Sigtia, our new model is a convolutional neural network (CNN). Due to the success of CNNs in the computer vision research field, its literature significantly influenced the music informatics research (MIR) community. In image processing literature, squared small CNNs filters (ie. 3x3 or 7x7) and square pooling are common. As a result of that, MIR researchers tend to use similar filter shape setups. However, note that the image processing filter dimensions have spatial meaning. Filters in image processing normally have small square filters to learn increasingly complex features. In the first layers, they learn small features like lines and curves. Then after pooling, which is a way of downsampling the feature maps or output from the convolutional layers, they perform convolution again with small filter shapes over the downsampled data. This is where the hierarchy of features come from, running the same size filters over data that become more and more downsampled, and obtaining

27

more abstract features like body parts and complex shapes.

Audio spectrograms filter dimensions correspond to time and frequency, so the dimensions need not be treated symmetrically. Intuitively, wider filters may be capable of learning longer temporal dependencies in the audio domain while higher filters may be capable of learning more spread timbral features. The Sigtia model uses unsymmetrical filters which is a start to handling audio data, but they merely try different shapes without justification. We believe that by using shapes that better model how each pitch is composed will produce better results.

The main change we test uses convolution filter sizes that span more of the frequency axis of the input spectrograms, instead of just small segments of it. The rationale for this is that in multi-pitch estimation we need to model the pitch that for tonal instruments, like piano, can encompass the full frequency axis. Thus, we need to take into account not only the fundamental frequency, but also the harmonics. Pons et al.[25] experiments with different filter sizes for the task of ballroom music classification. They find benefits in using filter shapes that are more appropriately suited for the task at hand. Benanne et al.[8] found one dimensional convolutions on spectrograms to be the best way to handle frequency invariance for learning music features from spectrogram data.

Additionally, for both the models with these new changes, as well as the baseline model, we also want to employ better training practices. One of these better practices is using the ReLU activation functions. Presently, it is only recommended to use ReLU or variants thereof for training deep neural nets. This is because the ReLU function does not saturate easily, unlike the sigmoid and tanh functions, which were used in Sigtia et al. Based on these activations, it also recommended to initialize our trainable weights using the initialization scheme. We also would like to implement better regularization techniques, other than just drop out. We use batch normalization

after each convolution and fully connected layer. Finally, we also want to do some more hyper-parameter optimization that the previous work did not, or at least that was not discussed in their paper. We will especially focus on testing the effect of initial learning rate and its decay schedule, as this is normally a vital hyper-parameter in the training process.

See Table 4.1 for all intended combinations of configurations we propose to test. In this table, we order models that we want to test starting from the baseline. Data flows through these layers or operations from the top of the list to the bottom. Dashes show emission of either batch normalization or pooling so that the same layer types of each of the models are synced up. The parameters inside each Conv2D layer represents the number of, width, and height of the filter shapes. The number inside each dropout layer represents the fraction of activations or intermediate data nodes dropped. The parameters inside the Max Pooling function represent the factor to how much to downsample, which in this case is only performed on the frequency axis. Finally, the parameters inside the Dense operation represent how many neurons are in the layer.

The first change we test uses the baseline with what is considered good practices: the ReLU function instead of the tanh/sigmoid combination between weight layers (paired with proper weight initilization) as well as batch normalization. We lower the dropout rates after convolutional layers because the convolutional layers do not have many parameters and overfitting is not as much of a problem, thus dropout does not have much effect. However, some dropout in the lower layers helps deal with noisy inputs and prevents overfitting in the later fully connected layers[33].

After verifying that these practices actually improves our results, we test a convolutional layer with intuitive filter shapes for handing spectrogram data. We first try to model how each fundamental frequency of each note in an octave changes over
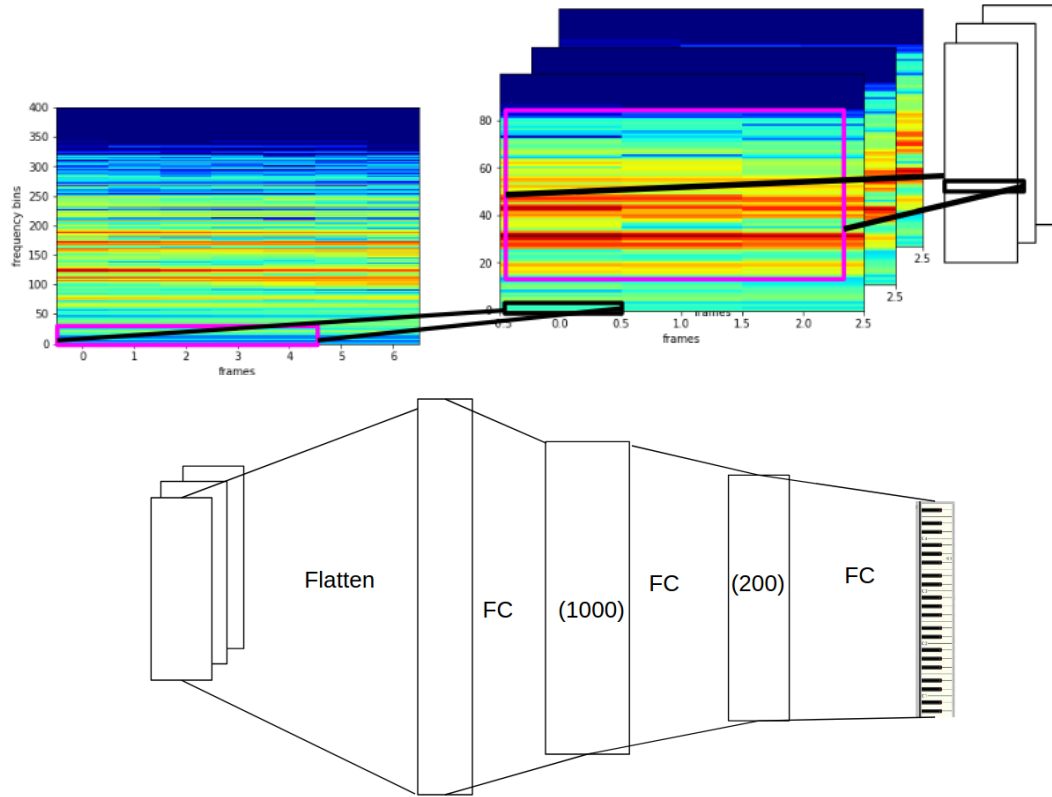
Figure 4.1: Model Visualization with new filter sizes, CNN (top) and Fully connected (bottom)

time. The first convolutional layer has filter shapes of $window\_size/2 * k$ where k is the number of bins per pitch (4). We also choose to stride vertically by k, because we are not concerned about the overlap between adjacent bins for different pitches. Then after downsampling, we use an actvation volume where the height represents the number of notes we are trying to predict. We then choose filter shapes that stretch vertically over the frequency axis (fs in the table) in an attempt to model pitch with each fundamental frequency and its corresponding harmonics. We test vertical heights of 36, 48 and 60 which encompasses 3, 4 and 5 octaves worth of pitch information respectively. We show a visual depiction of this model with a vertical height of 60 in Figure 4.1.

Because we are dealing with a multi-label problem, we use the binary cross-entropy as a loss function. This is described as follows:

Table 4.1: Different Model Configurations (Data forward propagates from top to bottom)

Model	Baseline	BaselineGP	NoteCNN
Stage 1	Conv2D(50x3x25)	Conv2D(50x3x25)	Conv2D(32x5x4)
-	-	BatchNorm	BatchNorm
-	Tanh	ReLU	ReLU
-	Dropout(0.5)	Dropout(0.2)	Dropout(0.2)
-	MaxPool(1x3)	MaxPool(1x3)	-
Stage 2	Conv2D(50x3x5)	Conv2D(50x3x5)	Conv2D(32x3xfs)
-	-	BatchNorm	BatchNorm
-	Tanh	ReLU	ReLU
-	Dropout(0.5)	Dropout(0.2)	Dropout(0.2)
-	MaxPool(1,3)	MaxPool(1x3)	MaxPool(1,2)
Stage 3	Dense(1000)	Dense(1000)	Dense(1000)
-	-	BatchNorm	BatchNorm
-	Sigmoid	ReLU	ReLU
-	Dropout(0.5)	Dropout(0.5)	Dropout(0.5)
Stage 4	Dense(200)	Dense(200)	Dense(200)
-	-	BatchNorm	BatchNorm
-	Sigmoid	ReLU	ReLU
-	Dropout(0.5)	Dropout(0.5)	Dropout(0.5)
Stage 5	Dense(88)	Dense(88)	Dense(88)
-	Sigmoid	Sigmoid	Sigmoid
Param #	2,012,738	2,013,138	2,109,992

$$L = \frac{1}{N} \sum_{n=0}^{N} y_n \log \hat{y_n} + (1 - y_n) \log(1 - \hat{y_n})$$

where N represents the total number of labels to be predicted. This loss function is commonly used for multi-label problems as it will allow us to measure loss between the predicted and ground truth (correct) vectors with multiple active labels.

We use the Keras API[6] combined with the TensorFlow backend[1], to easily implement both the baseline and proposed model as quickly as possible.

4.3   Postprocessing

While Sigtia et al. put much focus in using a recurrent neural network in their music language model, they did not see significant improvements in their result metrics compared to just using thresholds and hidden markov models. We will only use thresholds as a post-processing method for comparison. We use the same threshold for each pitch class to determine whether or not it is active. We find this threshold by calculating the precision-recall curve for potential thresholds across all notes and choosing one that maximizes both (we discuss these metrics more in the next chapter).

4.4   Piano Roll Conversion

Result: MIDI Sequence of Notes

pianoroll, fs = get_inputs()

changes = nonzero(diff(pianoroll))

note_sequence = []

for *frame,pitch in changes* **do**
    change = pianoroll[pitch,frame+1]

    time = frame/fs

    if *change == 1* **then**
      | note_start[pitch] = time

    end

    else
      note = create_MIDI_note(pitch,start=note_start[pitch],

      end=time,velocity=100)

      note_sequence.append(note)

    end

end

Algorithm 1: Piano Roll to MIDI

Once we have post-processed the model output scores into binary vectors indicating when notes are active, we then look at the change activity to indicate note on/off events. Using this information, we can accurately create MIDI file using the pretty_midi library using the MIDI note number, note on time, and note off time. We will only use a constant velocity value, though we could use the StyleRNN system[22] to get more humanly accurate velocity values, if transcription being 're-performed' in a serious way is desired. We also choose to do some further post-processing by setting a minimum time a note needs to be active for. We set it to 0.1s. See Algorithm 1 for pseudo-code to get some perspective of how this was accomplished.

Chapter 5

EVALUATION

5.1 MAPS Dataset

For the initial training and testing of our models, we first use a dataset of several classical piano MIDI files collected from www.piano-midi.de/. If we can overfit to these songs, we are reasonably certain that our models are set up well.

The main dataset used for evaluation of the proposed models is the music sections of MAPS dataset, which include 270 full recordings of 240 unique piano songs, 210 synthesized and 60 recorded on a real piano in 9 different ways. See Figure 5.1 to see a histogram of note counts across the dataset. From the histogram, we see that there are roughly 50,000 examples of each of the notes in the range from 40 (E2) to 80 (G#5). Across the entire music section dataset, we find 2,808,579 examples of 76,398 different combinations.

5.2 Training

To train effectively, we set up a pipeline of pre-processing, training, post-processing, and evaluation to be done easily from a master file of model configurations. This allowed us to set up training once and wait for results. All training was performed on a personal workstation with an Intel i7-7700K CPU, NVIDIA 1080Ti Graphics Card, and 32 GB DDR4 RAM.

To fully train each model, similar to Sigtia et al, we use the Configuration 2 of their paper, which trains on synthesized recordings and tests on real recordings. We use this configuration to realistically emulate the situation as labelled acoustic audio
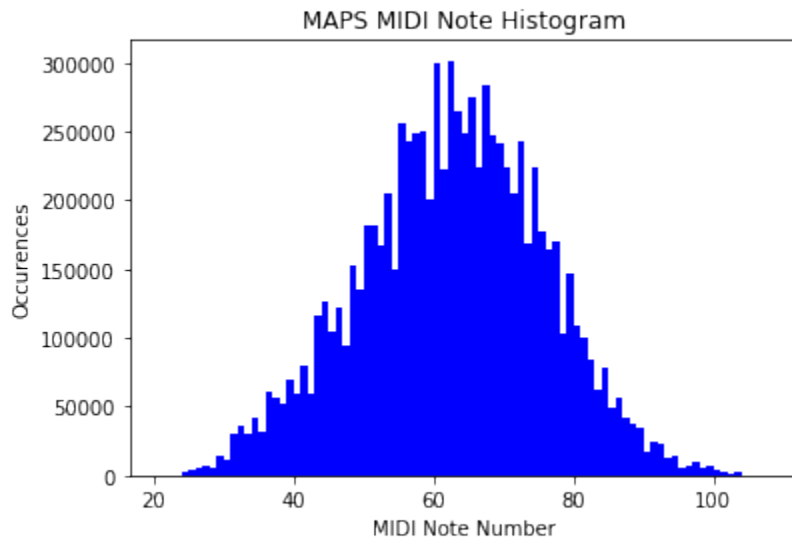
Figure 5.1: MAPS Dataset Histogram

is hard to come by. In this situation, we will train a model on synthesized music that is rendered in the sound of an instrument and then tested on the real recordings of the same instrument. The training split from the 270 total recordings is 180 recordings for training, 30 for validation, and 60 for testing.

We train for 1000 epochs and stop training early if the loss on the validation data does not improve for 20 epochs. For all models other than the baseline, we use an initial learning rate of 1e-3 with a learning rate decay schedule that halves the learning rate if the validation loss has not decreased for 5 epochs. For the baseline model, we use the same parameters as Sigtia: initial learning rate of 1e-2 decreasing linearly to 0 over 1000 epochs.

We train several models attempting to improve upon the baseline. The first is the BaselineGP which has mostly the same architecture as the baseline but uses the good practices we mentioned in the previous chapter. The rest of the new models (starting with the prefix NoteCNN) test using longer filter shapes. The number x at the end of each NoteCNN-x model represents how far the filters span over the frequency axis

for the second convolution, as discussed in the previous chapter. We test 3 sizes to determine how including more information across the frequency axis improves the results.

See Table 5.1 for the training time results associated with training each model. We see higher computation time for the new Conv2D model as it introduces batch normalization four times. The time per epoch increased for the BaselineGP model as the new layers added significant computation. However, the proposed NoteCNN model decreased computation time, for the second convolution layer spanned far enough across the frequency axis to greatly reduce the number of dot products between filters and local regions of the feature maps. All models seem to converge around the same time.

See Figure 5.2 for a comparison of training/validation loss models compared to the baseline model. In the chart for implementing good practices, we see much better training as the proposed changes converge to a lower loss and the model does not seem overfit as much. The Sigtia paper did not mention any other form of regularization than dropout, so we assume that dropout was the only method employed. However, other methods may have also been used. We see that comparing the good practices baseline with NoteCNN model, we manage to get some slightly better convergence in less epochs trained.

5.3 Testing

5.3.1 Metrics

Normally in machine learning problems, the common metric used is mean class accuracy. In our case, mean class accuracy doesn't work very well as our data is imbalanced and we are dealing with a multi-label problem. If we were to use mean class accuracy
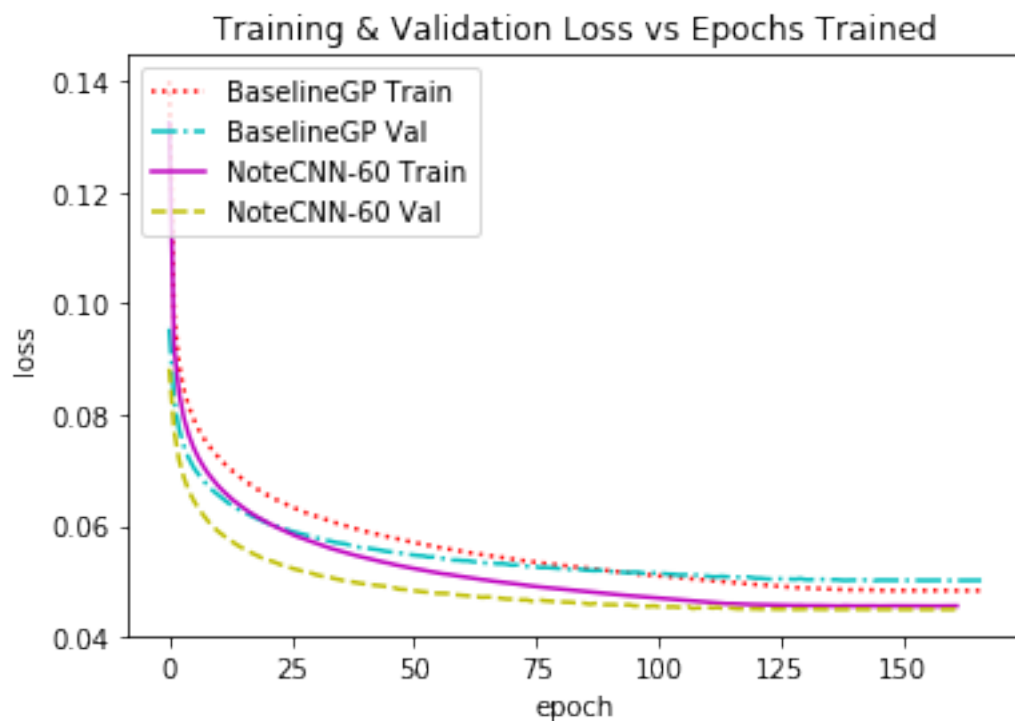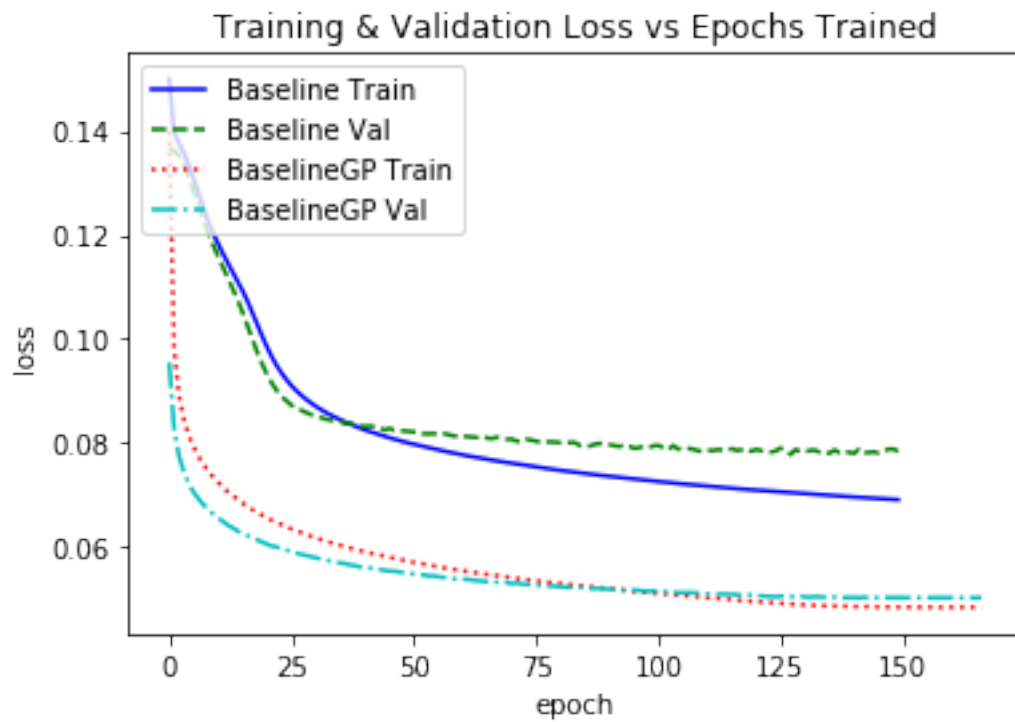
Figure 5.2: Model Loss vs Epochs Trained

Table 5.1: Time for Training each Model

model	total time trained	num epochs	avg epoch time
Baseline	5.385hrs	150	129.24s
BaselineGP	7.332hrs	166	159.024s
NoteCNN-36	4.5112hrs	181	89.7261s
NoteCNN-48	4.207hrs	171	88.5706s
NoteCNN-60	3.698hrs	162	82.1831s

metric, a model that always guessed zero would have a very high score. This is only because there are an average of 4 notes active while the other 84 notes are off.

Instead we use precision, recall and F1-score (harmonic mean of the first two). These are given by the equations:

$$P = \frac{\sum_T TP[t]}{\sum_T TP[t] + FP[t]}$$

$$R = \frac{\sum_T TP[t]}{\sum_T TP[t] + FN[t]}$$

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

where TP, FP, and FN stand for true positive, false positive, and false negative classifications. We use sklearn[24] to compute the global precision, recall, and fscore for all labels. We also compute these metrics for each label, to see how the system performs based on which note.

We first use these metrics for determining the optimal threshold for post-processing our predictions, as discussed in the previous chapter. We choose the thresholds that result in the maximum F1-score when the network output or scores and ground truth

are compared across the training and validation data. We then use this same threshold for evaluation over the testing data.

Automatic Music Transcription systems are normally evaluated on a per note basis. However, the Sigtia paper introduced frame-wise evaluation, as each example that goes through the network is for predicting which notes are in the center frame. Their results seemed similar for evaluations.

We also attempt to evaluate our system frame-wise. For evaluating note-wise, we use the mir_eval[27] library which evaluates music transcription systems by checking if the correct note events occur within 50ms of the ground truth.

5.3.2 Framewise Evaluation

See Table 5.2 for a comparison of test results between each of the models. Employing good practices increases F1-score by about 3%. The new proposed model increases F1-score by around 6% for the best performing NoteCNN model, showing that tailoring good filter shapes sizes to fit the data is important. It is also interesting to see how the metrics are affected by the improvements. We see that precision increases sharply while recall decreases for the best F1 score in the model only employing the good practices. In our best performing model, we see that precision gets boosted again and the recall nearly equals the baseline model.

In Table 5.3, we measure the frame accuracy of each system to see the percentage of frames where every note is guessed correctly across the test set. We also measure how long each system could correctly predict all notes in the frames. The accuracy seems to back up the same results from measuring the F1 scores across the dataset, but we see that the BaselineGP model actually gets the frames correct for the longest amount of time, though all streaks were around 5.5 to 5.9 seconds.

In Table 5.4, we test our metrics based on the amount of notes present in the

Table 5.2: Model Results for Framewise Evaluation on Test Data

model	P	R	F1
Baseline	0.61243	0.69352	0.65046
BaselineGP	0.69073	0.66101	0.67555
NoteCNN-36	0.72170	0.6953	0.70828
NoteCNN-48	0.7208	0.69986	0.71021
NoteCNN-60	0.72088	0.70163	0.71112

Table 5.3: Frame Accuracy and Max Streak times for each model

Model	Accuracy	Max Streak (Frames)	Max Streak (Seconds)
Baseline	0.0974	237	5.501
BaselineGP	0.2093	254	5.897
NoteCNN-36	0.2492	237	5.501
NoteCNN-48	0.2521	239	5.549
NoteCNN-60	0.2561	239	5.549

Table 5.4: F1 Score per Note Count for NoteCNN-60

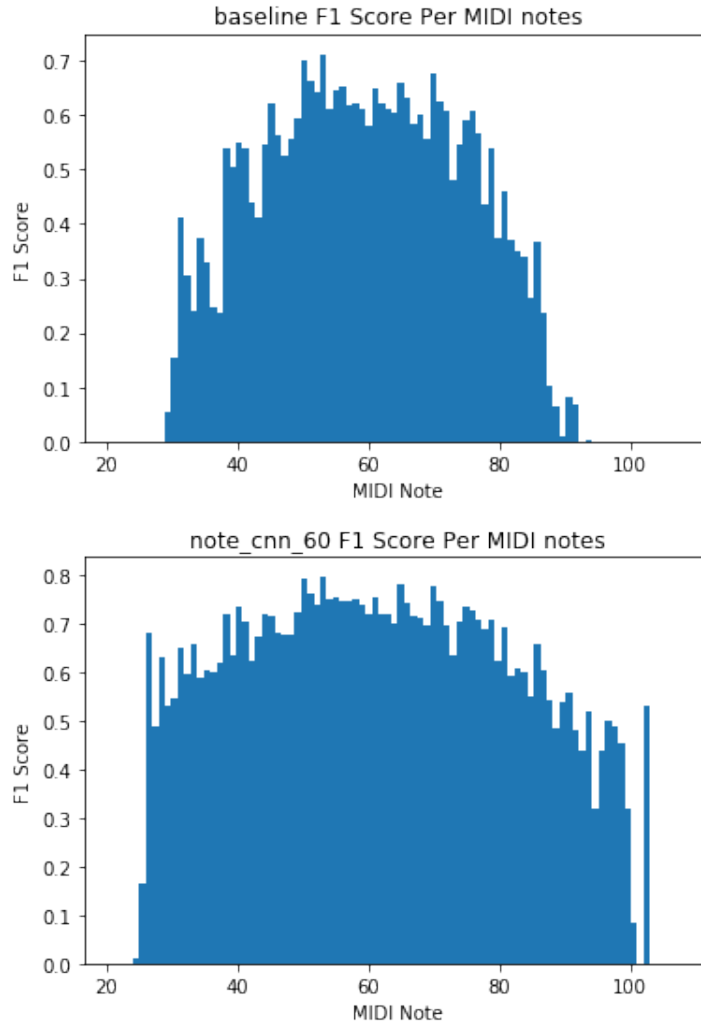| Note Count	# Examples	# Unique	P	R	F1
1	74330	79	0.4470	0.8339	0.5820
2	132819	1732	0.6002	0.7783	0.6777
3	132194	7063	0.7192	0.7344	0.7267
4	153187	10766	0.8068	0.7030	0.7513
5	80471	7608	0.8432	0.6621	0.7417
6	37080	4694	0.8756	0.6458	0.7434
7	8567	1916	0.8924	0.5822	0.7047

Figure 5.3: Model F1 Score vs MIDI note

frame for NoteCNN-60 model. It is also surprising to see that our F1 score is best when there are 4 notes active in a frame, though these F1 scores do seem to have a correlation to the number of unique examples.

In Figure 5.3, we look at F1 scores for each MIDI note for the baseline and the NoteCNN-60 model. We see that in our new model we achieve a higher overall F1 score. We see both models perform better for notes more in the middle range close to middle C (MIDI note 60). The baseline seems to match the bell curve shape of the histogram of note occurrences in Figure 5.1. We also find that our model is able

41

to pick up more of the less frequent notes, especially the higher ones. However, we still see a slight decline of F1 score as the note value increases from 80, which could mean that these notes are not common in the dataset, but also that we are unable to pick up the harmonics for the higher notes, based on the sampling rate.

5.3.3   Notewise Evaluation

We were unable to replicate Sigtia's results for notewise evaluation. We found that notewise evaluation directly depends on how one post-processes their system scores into binary predictions and then into a MIDI-like stream of notes. Sigtia et al. does not provide any further insight on how the conversion from binary piano roll to MIDI occurs, but we assume it is something similar to ours. When evaluated notewise, we see much lower results compared to framewise. For our best performing model, we see notewise evaluation F1 score under 15% and we attempt to explain some reasons why.

In Figure 5.4, we compare predictions for a stream of notes for a single pitch to ground truth, and see that some notes are predicted to toggle over the duration of time the ground truth note is active. Coalescing notes in this case seems to help in this example, but when tested over the full testing set, this seems to hurt the notewise evaluation results, possibly due to notes that do not need to be coalesced.

Another problem that contributes to the lack of performance in notewise evluation are spurious onsets incorrectly guessed by the system, but only for a small amount of frames. To deal with this, we set a minimum time threshold for how long a note should be active to be at least 50ms which is a little more than 2 frames. Setting this minimum time did seem to improve F1 score a bit.

We compare the notewise evaluation results for these extensions in Table 5.5. MT stands for minimum time and the parameter stands for the time in seconds. C stands
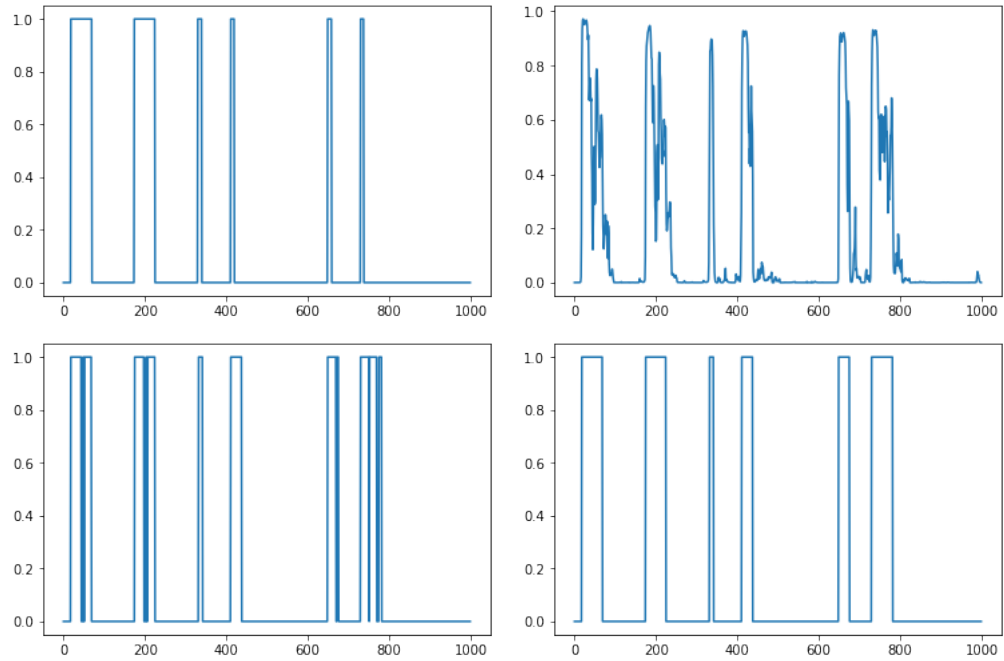
Figure 5.4: Example of Note coalescing in order from top left to bottom right: Ground Truth, Model Scores, Thresholding, Coalescing

for coalesce where the parameters stands for the amount of frames the max amount of frames between notes to coalesce together.

Table 5.5: Notewise Evaluation of NoteCNN-60 Model variants

model	P	R	F1
Baseline	0.07949274	0.13013225	0.09776747
NoteCNN-60	0.11657394	0.15982252	0.13375549
NoteCNN-60 w/ MT(0.05)	0.14331639	0.14926589	0.14484556
NoteCNN-60 w/ MT(0.1)	0.16130396	0.13218385	0.14218012
NoteCNN-60 w/ C(1)	0.11888187	0.14571494	0.12990134
NoteCNN-60 w/ C(2)	0.11717406	0.13190226	0.12303553
NoteCNN-60 w/ C(1),MT(0.05)	0.0.13979451	0.1389058	0.13804876

Chapter 6

CONCLUSION

In this chapter, we review the challenges faced, discuss possible future work, and give a final summary.

6.1 Challenges

There were many challenges in performing this research. The first was that the use of neural networks on audio data is nowhere near as common as using it on image data. Many computer vision projects have been released open source, which creates an easy starting point for research. However, most of this project had to be started from scratch, and much work was needed just to be able to re-implement the baseline model. This may be because there is a financial incentive in keeping these systems, especially music transcription systems, closed. We hope to see more open source projects in the audio processing community in the future. Another challenge was working with a large dataset. We had to set up significant resources for switching between data files and to handle computing statistics over the full training set in workable batches.

6.2 Future Work

There are many ways to improve this AMT system. The first way would be using information about the music key. By doing a first pass over the data and trying to segment the audio into segments with the same key would greatly improve the context in which multi-pitch estimation would occur. For each different key, there atr notes that would be much more likely to occur. Trying to determine the genre of the music

beforehand could also be of help, as in different genres of music, different scales of notes are used.

An interesting improvement to the system could also use filters with gaps. We can set up complex filters to be used in convolution that directly model the harmonic pattern of each note instead of using full rectangular shapes. It would be intriguing to see if excluding the rest of the information in the filter with gaps would greatly improve performance.

Another way in which this system could be improved would be to deal with various timing issues. Music follows a rythmic tempo and by quantizing the notes, you can structure the data to be more reflective with how music is actually written. The dataset we used did not have any ground truth recordings for actual human performances. Creating transcriptions for human performances could be more challenging to get quantized notes, as we would have to deal with the potentially major timing changes that occur when a human speeds up or slows down their playing. Another method could be a separate onset detection model in parallel with our end-to-end model that could help deal with the spurious onsets.

Additionally, there should be more research on transcribing music audio with different instruments. We only attempted to transcribe piano music, but since the start of this research there have been other music datasets released. One specific dataset of worthy mentioning is the MusicNet dataset[34], which is compromised of multi-instrument music audio performed by humans and aligned to MIDI-like data using dynamic time warping.

Finally, our system can only try to classify which notes are active but not how intensely. This leads to fairly inhuman-like music when the transcription is replayed. Future work may need to also use spectrogram data with transcription to determine each note volume, so a MIDI velocity value can be assigned.

6.3 Summary

We have presented a new automatic music transcription model that uses more intuitive filter shape sizes for handling spectrogram data. We believe that by choosing filter shapes more specifically designed for audio data, we see a significant improvement in performance and speed to train and transcribe.

BIBLIOGRAPHY

[1]   M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S.
      Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp,
      G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg,
      D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens,
      B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan,
      F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and
      X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems,
      2015. Software available from tensorflow.org.

[2]   S. Böck and G. Widmer. Maximum filter vibrato suppression for onset
      detection. In *Proc. of the 16th Int. Conf. on Digital Audio Effects (DAFx).
      Maynooth, Ireland (Sept 2013)*, 2013.

[3]   N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Audio chord
      recognition with recurrent neural networks. In *ISMIR*, pages 335–340, 2013.

[4]   J. C. Brown. Calculation of a constant q spectral transform. *The Journal of
      the Acoustical Society of America*, 89(1):425–434, 1991.

[5]   K. Choi, G. Fazekas, and M. Sandler. Explaining deep convolutional neural
      networks on music classification. *arXiv preprint arXiv:1607.02444*, 2016.

[6]   F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[7]   A. De Cheveigné and H. Kawahara. Yin, a fundamental frequency estimator
      for speech and music. *The Journal of the Acoustical Society of America*,
      111(4):1917–1930, 2002.

[8]   S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6964–6968. IEEE, 2014.

[9]   C. Dodge and T. A. Jerse. *Computer music: synthesis, composition and performance.* Macmillan Library Reference, 1997.

[10] V. Emiya, R. Badeau, and B. David. Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1643–1654, 2010.

[11] A. Guenther. Cal Poly Github. `http://www.github.com/CalPoly`.

[12] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[16] A. Kaparthy and F.-F. Li. Convolutional neural networks (cnns / convnets), 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[18] S. Lau. Learning rate schedules and adaptive learning rate methods for deep learning, 2017.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] C.-Y. Liang, L. Su, Y.-H. Yang, and H.-M. Lin. Musical offset detection of pitched instruments: The case of violin. In *ISMIR*, pages 281–287, 2015.

[21] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[22] I. Malik and C. H. Ek. Neural translation of musical style. *arXiv preprint arXiv:1708.03535*, 2017.

[23] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] J. Pons, T. Lidy, and X. Serra. Experimenting with musically motivated convolutional neural networks. In *Content-Based Multimedia Indexing (CBMI), 2016 14th International Workshop on*, pages 1–6. IEEE, 2016.

[26] C. Raffel and D. P. Ellis. Intuitive analysis, creation and manipulation of midi data with pretty midi.

[27] C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, D. P. Ellis, and C. C. Raffel. mir_eval: A transparent implementation of common mir metrics. In *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*. Citeseer, 2014.

[28] D. Ross. If it wasn't for mozart, this piece would have been forgotten forever, Sep 2013.

[29] E. D. Scheirer, L. Vercoe, W. Picard, and E. D. Scheirer. Music perception systems. 1998.

[30] 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 (TASLP)*, 24(5):927–939, 2016.

[31] P. Simon. *Too Big to Ignore: The Business Case for Big Data*. Wiley and SAS Business Series. Wiley, 2013.

[32] P. Smaragdis and J. C. Brown. Non-negative matrix factorization for polyphonic music transcription. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 177–180. IEEE, 2003.

[33] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[34] J. Thickstun, Z. Harchaoui, and S. Kakade. Learning features of music from scratch. *arXiv preprint arXiv:1611.09827*, 2016.

[35] E. Vincent, N. Bertin, and R. Badeau. Adaptive harmonic spectral decomposition for multiple pitch estimation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):528–537, 2010.

[36] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2285–2294, 2016.