Revealing the Determinants of Acoustic Aesthetic Judgment Through Algorithmic "Dreaming"

Spencer D. Jenkins

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Benjamin C. Jantzen, Chair

Bert Huang

Sang Won Lee

May 14, 2019

Blacksburg, Virginia

Keywords: Deep Learning, Machine Learning, Philosophy of Aesthetics

Revealing the Determinants of Acoustic Aesthetic Judgment Through Algorithmic "Dreaming"

Spencer D. Jenkins

(ABSTRACT)

This project represents an important first step in determining the fundamental aesthetically relevant features of sound. Though there has been much effort in revealing the features learned by a deep neural network (DNN) trained on visual data, little effort in applying these techniques to a network trained on audio data has been performed. Importantly, these efforts in the audio domain often impose strong biases about relevant features (e.g., musical structure). In this project, a DNN is trained to mimic the acoustic aesthetic judgment of a professional composer. A unique corpus of sounds and corresponding professional aesthetic judgments is leveraged for this purpose. By applying a variation of Google's "DeepDream" algorithm to this trained DNN, and limiting the assumptions introduced, we can begin to listen to and examine the features of sound fundamental for aesthetic judgment.

Revealing the Determinants of Acoustic Aesthetic Judgment Through Algorithmic "Dreaming"

Spencer D. Jenkins

(GENERAL AUDIENCE ABSTRACT)

The question of what makes a sound aesthetically "interesting" is of great importance to many, including biologists, philosophers of aesthetics, and musicians. This project serves as an important first step in determining the fundamental aesthetically relevant features of sound. First, a computer is trained to mimic the aesthetic judgments of a professional composer; if the composer would deem a sound "interesting," then so would the computer. During this training, the computer learns for itself what features of sound are important for this classification. Then, a variation of Google's "DeepDream" algorithm is applied to allow these learned features to be heard. By carefully considering the manner in which the computer is trained, this algorithmic "dreaming" allows us to begin to hear aesthetically salient features of sound.

Contents

Chapter 1

Introduction

There are two main components to this project, each beginning to answer an important question in the study of acoustic aesthetics. The first component seeks to determine what is interesting to a professional composer. To achieve this, a deep neural network (DNN) capable of replicating acoustic aesthetic judgments was developed. This DNN was trained to imitate the preferences of Dr. Eric Lyon, a professor in the Music Department at Virginia Polytechnic Institute and State University, by leveraging a unique corpus of sounds he developed. As a component of his creative process, Dr. Lyon works with found sounds or simple computer-generated waveforms. He modifies these sounds by applying a variety of transformation algorithms to them. Only the sounds that he judges interesting are chosen for further transformation or for inclusion in his compositions. This forms a cyclic, iterative process for creating interesting sounds by switching between generation and judgment. The rating given to a sound takes the simple form of a bivalent label: "interesting" or "not interesting." Thus, Dr. Lyon's creative process affords us a unique dataset consisting of a large collection of acoustic waveforms with corresponding professional aesthetic judgments on their interestingness. This is the dataset with which our network was trained to perform the following binary classification problem: determination of a given input sound as "interesting" or "not interesting."

In addition to allowing for the "dreaming" components of this project described below, this ability to perform binary classification is helpful in its own right. Currently, Dr. Lyon must

sift through every sound generated through his algorithmic process, listen to it, and assign it an aesthetic judgment. Only after this is completed can the next round of transformations be applied. By allowing our DNN to perform automatic classification of the sound, we can streamline this process tremendously. We can produce multiple rounds of generation at the push of a button without any intervention by Dr. Lyon. By learning the aesthetic preferences of Dr. Lyon during its training phase, the DNN is capable of determining which sounds it receives as input are interesting and thus worthy of continued transformation or selection.

The second major component of this project attempts to answer the following question: what are the elemental features responsible for a sound being interesting? We explore this by modifying Google's "DeepDream" algorithm [19] and applying it to our trained network. Initially, [19] was developed to investigate networks trained for image classification tasks. It allowed for the probing of the inner representations learned by the network, enabling visualizations of these learned features. [19] achieves this by inverting the traditional method of training a classifier. Instead of modifying the weights of the network, we rather modify an image input to the network via gradient ascent. We do this to maximize the activation of given filters learned in the network. Starting with an image consisting of random noise, we can iteratively tweak this image into an image which cause specified filters to activate maximally. Thus, the optimized image represents what these filters are "looking" for. We can do the same thing for our network; however, instead of visualizing the features of an image for which the network is "looking" to determine its classification, we instead hope to audiate the features for which the network is "listening" to determine its aesthetic judgment. This means that we can hear aesthetically salient features of sound, based on the preferences on which the network was trained.

Figure 1.1 illustrates how this process was used for visual data in the original DeepDream effort [19]. As one can see, the filters over which the input image was optimized learned

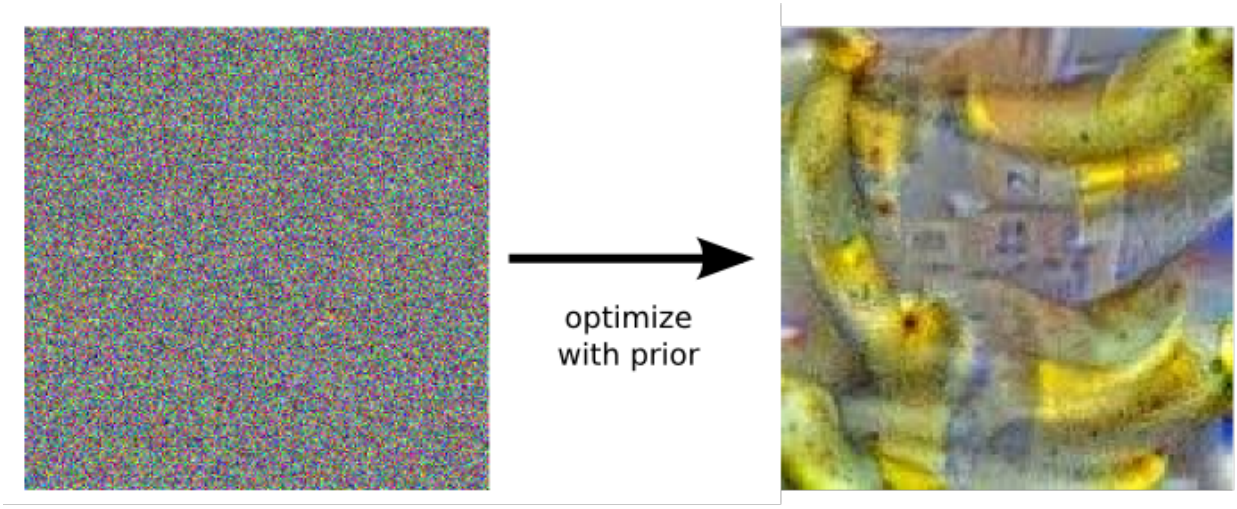

Figure 1.1: Features for detecting a banana [19]

representations useful for detecting bananas in an image.

There is also another, very similar use of the DeepDream algorithm with an important difference. Instead of starting our optimization on a random noise image, we instead optimize on an actual image. When we perform gradient ascent on an image to maximize the activation of a given filter, we are boosting those features in the image which the filters represent. Figure 1.2 is an example of this technique from the original DeepDream paper [19]. The original image (on the left) has been optimized (on the right) so as to maximize the activations of the network filters trained to detect strokes and edges.

When we apply the same approach to audio data, we can boost the features in the audio that have been determined to be aesthetically salient by the network. The ability to hear these relevant features emphasized in a sound is of great explanatory interest for aesthetics. Ideally, we hope to discover either some tonal or metrical structure in these learned features.

For both of these project components, we would like the features that our network learns (and that we visualize) to be as aesthetically fundamental as possible. In other words, we would like to avoid smuggling in assumptions about aesthetically "interesting" sound

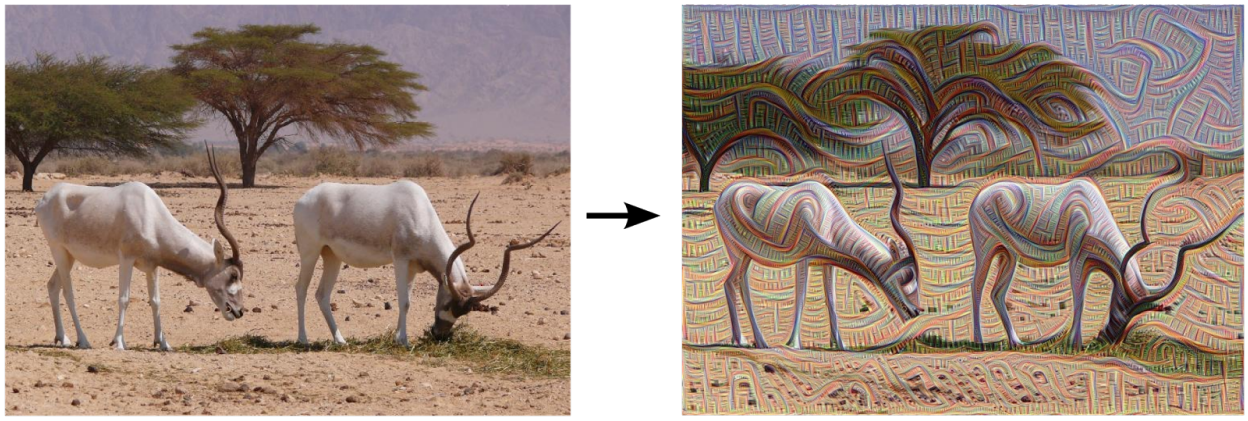

Figure 1.2: Features for detecting edges and strokes [19]

into our model. Ultimately, we would like the network to determine for itself what exactly these features may be. As will be discussed in the review of related works, many current classification approaches for audio data contain assumptions about the structure of both the data and the relevant features contained therein. One of the major goals of this project is to use a DNN to help us interpret the features that constitute an aesthetically interesting sound. Thus, the incorporation of assumptions begins to beg the question about the very elements we wish to uncover through this project. Even if the addition of these biases increases performance of the network (by lowering the loss of the network on the training data, or increasing the value of the network's performance metrics such as precision, recall, etc.), we would prefer to obtain a model without using them. The fundamentality of these assumptions for aesthetic classification is exactly what is in question. Choices made during the design of the network impose biases on the types of features that the network will learn, thus affecting our study into what aspects of sound are aesthetically fundamental. Many examples of these decisions will be explained in the following sections, with discussion into the ways our decided approaches mitigate any biases introduced.

Chapter 2

Dataset Description

The sounds generated by Dr. Lyon, with which we trained our network, come from a previous work of Dr. Lyon's: Mushroom. Mushroom is what is known as an oracular processor. These perform operations on an input sound to create a new, modified sound. An oracular processor gains its name from the fact that the user has very little parametric control over the generation process (or no control at all); the processor acts as an oracle, deciding how to modify the sound on its own. An oracular processor is useful in promoting new, exploratory sounds. By giving the computer control over the parameters of transformation, the user can avoid retreading familiar and enticing sequences of transformations. The process implemented with Mushroom is well described in [1] and [18], but I will provide a summary of the main points below.

First, as previously mentioned, Mushroom makes its parametric decisions internally without input from the user. Upon being presented with an input sound, Mushroom performs a randomly selected series of processes to create an output sound. This is repeated for the specified number of output sounds the user requested. Mushroom also allows for multiple levels of processing to occur. The input sound presented to Mushroom is said to be at level 0. The sounds produce by running an iteration of Mushroom on the level 0 sound are at level 1. To produce a level 2 sound, a level 1 sound is randomly chosen (from the produced level 1 sounds) to undergo a random series of processes. This same procedure occurs to produce sounds of even higher levels, using a random selection from the sounds at the previous level.

Thus, sounds at higher levels "collectively embody complex sonic interrelations among all the sounds at the different levels." [18]

There are two additional features of Mushroom that are interesting, particularly in the context of this project. First, Mushroom contains a utility called Mushmimic. This allows a user to discover the sequence of processes applied to a sound of interest. This sequence can then be applied to new, different input sounds. Second, Mushroom also allows a user to specify sounds as unwanted; these are removed from the pool of sounds to be processed for the next level of output. For example, let's assume a user inputs five sounds to undergo two levels of processing. After the first level of processing occurs, the user decides that two of the transformed sounds are undesirable. Only the remaining three sounds are available to be chosen for the next level of oracular processing.

Both of these features, the selection of interesting sounds and the selection of undesirable sounds, require human intervention. The user must actively listen to and label these sounds as such. Our network, trained to mimic the aesthetic preferences of Dr. Lyon, allows for an automation of both of these processes. This is in keeping with the general philosophy of an oracular processor, giving the computer the control instead of the user. Additionally, it helps in removing the majority of the burden of the process from the user. The generation of new sounds using Mushroom is easy, but Dr. Lyon describes the process of listening and labeling these sounds as time-consuming. Not only does it require a large amount of time, but it is also necessary for Dr. Lyon to be in the right frame of mind to perform this creative task; in other words, it imposes a large burden on the user in an otherwise autonomous system. With the automation of judgment enabled by this project, an artist employing Mushroom can create a much larger collection of sounds, tuned to their aesthetic preference, with a great reduction in the effort required.

In addition to the library of sounds created using Mushroom, Dr. Lyon has also assigned a

judgment to each of the included sounds. This takes the form of a binary label indicating interestingness of the sound: 1 represents “interesting”, with 0 representing “not interesting.” These judgments form the labels our network was trained to predict.

Chapter 3

Related Work

3.1 Google's DeepDream

Originally titled "Inceptionism," Google's algorithm known today as "DeepDream" was introduced in a 2015 blog post [1]. It proposed optimizing over the input of a network, instead of the network's parameters, to understand the network's learned representations. This was not the first time that this general approach toward interpretability was taken. The 2009 paper "Visualizing Higher Layer Features of a Deep Network" from Erhan et al. applied this optimization idea to visualize the features learned by deep belief networks and auto-encoders trained on the MNIST dataset [2]. Though the main idea is the same, we are more interested in the work of Mordvintsev et al., as they applied this idea to convolutional neural networks (CNN). This allowed them to visualize the features that the filters of a CNN come to represent. This is exactly what we are trying to do, with the difference being that our network's inputs are not images but sounds. So, instead of visualizing the learned representations, our project allows us to listen to them.

At its core, the optimization process for image data is a simple one. We perform gradient ascent on an image with the goal of maximizing the responses of a given set of network filters. With each iteration of gradient ascent, our input slightly changes in a way that increases the activations of the specified filters. Given enough iterations, we can obtain an input that maximally activates the network features under consideration.

One reason that this is especially interesting in a CNN is due to the hierarchical feature representations learned by these types of networks. As one looks at deeper layers (further from the input) in a CNN, the learned filters respond to increasingly abstract features. Take, as an example, the network on which the DeepDream algorithm was initially proposed. This network was trained to perform classifications on the ImageNet dataset [6]. ImageNet is a dataset consisting of 3.2 million images spread across more than 5247 categories. Based on the hierarchical nature of WordNet [8], the categories form a treelike structure. Some higher-level categories in ImageNet include things such as animals, vehicles, foods, and many more. In the upper layers (closer to the input) of the neural network, the filters learned to represent basic structures. These take the form of such objects as lines and strokes. Deeper layers in the network perform their convolutions over the feature maps of the previous layers. Thus, the representations that these filters come to learn are at higher levels of abstraction, as they are built from the features learned by the previous layers. In a network trained on a dataset such as ImageNet, deeper layers of the network start to learn filters for detecting eyes, faces, wings, etc. Thus, by selecting different layers over which to "dream," we can see different complexities of features learned.

A common addition to the optimization approach of DeepDream is that of regularization. It is introduced in the attempt to make the resulting visualizations more interpretable. The regularization methods are typically inspired by the natural structure of images. For example, one might penalize a variance between neighboring pixels. Another common approach is to encourage visualizations that maintain the same high activations of filters under various transformations (e.g., rotations, scalings, and jitters). These help to discourage visualizations of adversarial examples, favoring natural looking results. For an excellent survey of these types of techniques, see the Distill paper "Feature Visualization." [20]

We choose not to employ these types of regularizations in our model for two reasons. First,

these are clearly designed to take advantage of the statistical properties of images. Thus, they are not well suited to the audio data we are using. Rotation invariance, for example, seems to be ill-suited for our data because audio features have a natural orientation in time (a sound played backwards is perceptually very distinct from the same played forwards).

Second, we are trying to ascertain features at as fundamental a level as we can. Though some of these regularization methods may produce better or more interpretable audiations, they may be less aesthetically relevant. This is in fact the question at hand; we are not sure whether invariance to any given transformation (however those may manifest in the audio domain) is important in the determination of quality. To introduce such techniques for the purpose of improved listenability starts to beg the question in regard to aesthetically salient features.

We do, however, share the goal of avoiding adversarial examples. This is tricky as they are less obvious in the audio domain than they are in the visual domain. Thus, the best that can be done is to perform some general regularization on the optimization. In order to avoid any bias toward the types of filters learned, we constrain our approaches to those that are physiologically motivated. By performing regularizations on the scale of physical imperceptibility, we can gain some benefits of regularization without sacrificing assumptions about the sound. As an example, we could perform smoothing on frequencies closer than the frequency resolution of the human ear. Regularization on this scale merely biases the results to those that are heard by humans. Any feature loss induced by this method would be imperceptible, meaning that we are not removing any aesthetically salient information from the signal. We discuss some of these ideas in the future work section of our conclusion.

3.2 Audio DeepDream

One of the closest efforts to our work is "Audio DeepDream: Optimizing Raw Audio with Convolutional Neural Networks," a 2016 effort by Google's Ardila et al [21]. Building off of the work of Google's DeepDream, the authors hoped to apply this same approach to audio. Toward this goal, they first trained a convolutional neural network (CNN) to predict 100-dimensional collaborative filter embeddings for 30-second clips of musical tracks. After the training phase was completed, gradient ascent was performed on audio input to the network (starting as either noise or silence) so as to maximize the activations of specified features in the network. The example the authors give is that of maximizing the mean activations of the last layer. Just as in the original DeepDream work, this allows us to gain a human-interpretable representation of the features that the network is learning. In this case, we can hear the sounds on which network features are trained to activate. Though the paper is short on details of implementation, there are several design decisions that the authors make. Our network differs in several of these, which allows us to determine more fundamental and aesthetically relevant features of sound.

First, our input data are not segments of music. Rather, they are algorithmically generated sounds. This means that any sort of musical structure present in Audio DeepDream's input is not present in the input to our network. Using music as input means that Audio DeepDream potentially learns feature representations of musical structure, or features on the timescale of musical structure. Because we are after the aesthetically relevant features of sound, not just specifically those of music, the data on which the networks is trained makes a large difference. The features we're interested in may in fact explain what makes music music in the first place!

Second, the classification task of Audio DeepDream is related to ours, but different in an

important way. While our network performs binary classification in the aesthetic categories of "interesting" or "not interesting," Audio DeepDream predicts a 100-dimensional collaborative filter embedding for its input [21]. This difference is subtle, but the Audio DeepDream approach starts to stray from the aesthetic questions we are interested in answering by essentially learning a proxy measure of aesthetic quality. This leads to results that are less robust and, in our opinion, less revealing than those that our network produces.

Collaborative filtering is a technique used in recommendation systems. On a basic level, recommendation systems consist of users and items. The goal is to determine the preferences of users toward items. These systems usually possess some knowledge about a user's preferences (e.g., a user's ratings of a subset of the items available in the system). A recommendation system hopes to determine how a user would rate items for which they have not expressed a preference. Collaborative filtering is one technique to do this, allowing a system to predict a user's preference toward certain items by leveraging the preferences of similar users. It does this by learning vector embeddings for both the users and the items on which the system operates. Abstractly, a user embedding encodes the preferences of a user and an item embedding encodes the features of an item. These embeddings can then be used to determine user A's rating of item B by performing a dot product between the embedding of user A and the embedding of item B. In the collaborative filtering system used for the data in the Audio DeepDream paper, these embeddings are learned using the method of Weighted Alternating Least Squares (WALS) [12]. WALS allows the embeddings for both the users and the items in the system to be learned jointly, with each learning iteration of the user embeddings informing the next learning iteration of the item embedding, and vice versa. By alternating our learning steps, we can learn both categories of embeddings at the same time.

Our binary classification approach is better suited for our goals than that of the collaborative filter embedding predictions. One benefit is the comparative robustness of our approach.

The embeddings learned in the Audio DeepDream approach are only meaningful with the corresponding user embeddings. A pair consisting of a user embedding and an item embedding is needed to make a prediction of aesthetic preference of a user toward that item. These learned item embeddings would not transfer to a system that has learned a different embedded representation of users. This is why we view this approach as learning a proxy measure for the aesthetic significance of features. It is dependent on the user embeddings of the CF system, embeddings which were already determined before the training of the network.

We also feel that training our network to perform binary classification on an aesthetic feature allows us to learn filters that are more closely connected with the corresponding aesthetics of the input sounds. The vector embedding of the CF approach is essentially a proxy representation of an aesthetic judgment, not the actual aesthetic judgment itself. Thus, any filters we learn in a CNN trained to predict these embeddings will inherit the biases (correct or incorrect) represented in the collaborative filter embedding. As one example, it is hard to determine the relative importance of the user embedding versus the track embedding as related to aesthetic preference. Both seemingly encode information necessary to make an aesthetic judgment on an input. What is not evident, however, is whether the aesthetically salient features of a sound are encoded solely (or even in majority) within the track embedding. It is entirely possible that the user embedding contains a good deal of information necessary for aesthetic judgment. Because we don't know the types of features that we are after, it's difficult to assign relative relevance to the two embeddings. Because we wish to influence our network as little as possible in the types of representations it learns, it seems a better approach to use the actual aesthetic judgment as the label.

Another difference between Audio DeepDream and our model is the inclusion of batch normalization [13]. Audio DeepDream includes batch normalization for every layer in the net-

work. In fact, they list batch normalization as a detail they found to be important to performance, calling it "a necessary addition." Batch normalization standardizes the inputs into each layer of a network. It does this by subtracting the batch activation mean from the layer input and dividing by the batch activation standard deviation. Assuming our training data is normally distributed, this already starts to make assumptions about the importance of features in the data. If our training data or layer activations are not normally distributed, then this type of operation can lead to strange results. Standardizing a lognormal distribution, for example, does not have the same effects as it does on a normal distribution; we don't obtain the same standardized distribution for every input lognormal distribution. We do in the case of the normal distribution, as standardizing provides us with a standard normal distribution, but we do not for many distributions.

Batch normalization also includes learned location and scale parameters that are applied after the standardization. This still does not remove the feature biases imposed by this method. In the case of different batches not belonging to a single location-scale family of distributions, this is especially problematic. This transformation causes us to learn network features about some further modified representation. Thus, in our quest to avoid biasing our network's representations of aesthetically salient features, we have opted not to include batch normalization. We are far more concerned with finding meaningful, fundamental features than we are in decreasing convergence time.

Google's Audio DeepDream mentions the need for regularization in the paper's conclusion by calling for "more advanced constraints in the optimization procedure." The simple regularization measures performed in the Audio DeepDream paper seem once again to not be motivated by physical auditory constraints. They propose the following methods to constrain their optimization procedure: 1) normalize the gradient by its maximum before taking an optimization step, 2) adding a sparsity constraint across the activations of the first layer,

and 3) adding a continuity cost on the first layer activations of neighboring frames. The first of these seems to be possibly aimed at decreasing the convergence time of the optimization, while the second two seem to be designed with some sort of regularization in mind. Methods 2 and 3 in particular seem to begin to bias the types of features the network begins to learn. Using physically inspired regularization techniques feels like a better way to constrain the optimization procedure, specifically when hoping to uncover aesthetically relevant components.

In the conclusion of the Audio DeepDream paper, the authors suggest that a possible future direction would be to include alternative perception tasks. This is precisely what we are attempting in our approach. By binary prediction of an aesthetic value, we are using the basic structure proposed in the paper to answer a different question about sound.

3.3 Networks Operating on Extracted Features

There are also several attempts that do not use raw audio to make predictions about sounds. These efforts first extract features from the sound before they are fed into a DNN. One such example is from "Audio-Based Music Classification with a Pretrained Convolutional Neural Network," a paper from Dieleman, Brakel, and Schrauwen [7]. In their work, they attempt multiple classification tasks on the Million Song Dataset; these include artist determination, genre determination, and key detection.

Classification tasks of this nature are not well suited for learning fundamental, aesthetically salient features of sound. They begin to sneak in assumptions about important features into our model, directing the representations toward those that are musically relevant. But, more importantly, this paper also imposes assumptions on learned features with the inputs it uses to train the network.

Instead of using the raw audio files as the network's input, features are first extracted to form the input. In this paper, they take the form of a 24-dimensional vector; 12 of these dimensions represent various timbre features, while the other 12 dimensions represent chroma components (relative presence of pitches). These vectors are computed for multiple segments of each audio track in the dataset. According to the authors, these segments "roughly correspond to the onset of notes or other musical events." This type of input already assumes a lot about the elements of musical perception or understanding. Part of what's at issue, however, is the fundamentality of these features. By using unsegmented raw audio as the input to our network, we hope to learn the most fundamental representations necessary for the determination of aesthetic quality without the bias of preconceived notions.

Another common approach for feature extraction can be seen in "Deep content-based music recommendation," a paper from van den Oord et al. [23]. In this paper, they attempt to use a CNN to predict latent factor vectors for audio tracks from the Million Song Dataset. These latent factor vectors are used in a recommendation system, similar to those that are learned in Audio DeepDream. The Million Song Dataset does not contain latent factor vectors, as it is not a part of a recommendation system but rather a collection of metadata about audio tracks. Thus, the authors formed their own ground truth vectors by combining user play count data from other sources. Essentially, they make the assumption that someone who listens to a song many times enjoys that song. Already, we can see how this approach might begin to bias the feature representations learned. The feature extraction performed on the audio, however, is a major difference from our approach.

First, the sound is split into windows of 3 seconds. The prediction for the entire audio track is determined by averaging the predictions over consecutive windows. This sort of segmentation is a common approach; as the paper mentions, this is done to speed up training time. As we discuss in the next section, however, this splitting begins to impose assumptions of structure

that we would like to avoid. For each of these segments, a spectrogram is computed. This spectrogram is what is fed into the network as input. A spectrogram is a time-frequency representation of a signal, with the horizontal axis representing time and the vertical axis representing frequency. This allows us to represent the magnitude of frequencies in a signal at a given time. At first glance, this seems to be a reasonable input representation to use. While some discretization needs to occur in determining the frequencies and times to display in the spectrogram, these decisions can be informed by physical considerations. The major reason that we stray from this approach is invertibility. A major portion of our project is being able to listen to sounds that maximally activate learned features in the network. If our input is spectrograms, the "dreaming" optimization process will produce spectrograms. In order to listen to these, we need to convert them back into an audio signal. There are iterative methods for doing this, such as the Griffin-Lim algorithm [10]. Because of their iterative nature, however, these methods leave behind artifacts in the reconstruction.

There are two reasons that this is troubling for our goal. First, we wish to uncover aesthetic features in sound and be able to hear them. If these sounds contain artifacts from an inversion process, it will be difficult to discern what is actually aesthetically relevant and what is merely a leftover from the inversion technique used. Second, we would like to use our "dreaming" process to transform sounds in a way that highlights aesthetically salient features. If this transformation introduces artifacts into the sound, it will not be as useful as an artistic tool. We would prefer our transformation to be solely based on the aesthetic preferences on which the network was trained. These two reasons are why we choose raw audio, rather than a spectrogram representation of the signal, as the input to our network.

3.4 Sample-Level Deep CNN For Music Auto Tagging

Another closely related work is that of "Sample-Level Deep Convolutional Neural Networks for Music Auto-Tagging Using Raw Waveforms," a 2017 paper by Lee et al. [16]. This is an important work because it takes raw audio as input to the network. This is in contrast to some of the networks described above, which take extracted features as input. From the raw audio, the network attempts to predict tags for the given sound. The sounds on which the network is trained (and the tags being predicted by the network) come from the Million Song Dataset and the MagnaTagATune dataset [3] [2]. The tags present in these datasets are abstracted features about the song, such as loudness, danceability, and time signature. A complete list of the tags contained in the datasets can be found in the references.

This work is similar to ours through the use of raw audio as input, meaning no assumptions are imposed concerning the structure of sound. In fact, as the title alludes, Lee uses far smaller convolutional filters than our network (e.g., filters with widths of 3 samples). The end goal of this network, however, is much different. In the attempt to predict the types of tags that the datasets in question use, the network starts to learn features that are useful for the classification of music. The tags are of a highly abstracted nature, reliant on theory-heavy notions of genre and musical structure. In determining the aesthetic qualities of sound, we would like learn features that are of a more fundamental nature. Though there is surely some overlap, particularly in features at the lower (closer to the input) layers of the network, the types of features learned to assign a tag of "rock" or "jazz" are already more abstracted than those learned to assign binary aesthetic valence to a sound.

Another important distinction is the size of the inputs fed to the network. In our approach, we submit a sound in its entirety as input to the network. The filters learned by the network are convolved over the entire input. Lee follows a similar approach as that used in van den

Oord et al. with regard to input size. Instead of feeding the entire sound into the network, the input is split into segments on the order of a few seconds (i.e., 1 to 4 seconds). The network then makes predictions on these truncations. To determine the prediction for the entire sound, the predictions on the segments are averaged to form a prediction on the whole audio track. This approach might make the classification task more computationally tractable, but it begins to bias the types of features learned by the network. First, it begins to assume the scale on which relevant features appear, as information of importance is constrained to the length of the segments created. Second, the positioning of these segments can be viewed as arbitrary in the sense that they occur at a specified periodicity. What if they end up splitting an important segment of sound in two, with neither half being able to trigger a response? Even applying the method of averaging over the segment predictions would miss out on this feature. Thus, you begin to bias the features learned toward those on a small scale without long-term dependencies. It may well be the case that this space is where aesthetically salient features live; however, we would rather have the network learn that this is (or is not) the case on its own, rather than presupposing it. This is why we choose to use the entire segment of sound as input to the network. While it may increase the computational needs of the system by increasing the size of network, it does so in a way that does not smuggle assumptions about useful features into the model.

One area where this type of segmentation approach seems particularly problematic is in regard to rhythmic importance. One of the most fundamental aspects of musical cognition is that of rhythmic perception. In [11], the authors make the case for rhythmic cognition being of particular importance to the study of musicality. When describing how the study of the origins of music should be conducted, they lay out five constraints that should guide such accounts. One of these constraints is entrainment, the ability to synchronize actions. By showing the ability for this behavior across many species, the authors make the case that

this is a fundamental component of musicality; at the very least, it is a component that needs to be considered when studying musicality. By averaging predictions over segments of audio, you begin to lose any long-term dependencies present in the data. This would begin to rule out the importance of rhythmic structure in the determination of aesthetic quality. By giving the network the entire unsegmented sound as input, it is allowed to determine for itself the importance of rhythmic structures as related to aesthetic valuation.

Chapter 4

Methods

The dataset provided by Dr. Lyon for this project consisted of 998 sounds, each with a corresponding label of "interesting" or "not interesting." These sounds were created using the previously described system of Mushroom; the initial input to Mushroom consisted of a field recording of a trombone being played. There is a wide variety in the types of sounds represented within the dataset, with many levels of processing represented; there are sounds consisting of 1, 2, 3, and 4 levels of processing applied by Mushroom present in the dataset. Each song was approximately 10 seconds long. One of the variable features in Mushroom is the length of the output sound, so not every training sample was exactly 10 seconds in length. Because of the fixed size input of our CNN, however, we needed each training sample to be of the same length. Thus, we used the command line utility SoX [4] to trim each of the supplied sounds to 10 seconds in length. In addition, we also introduced a half-second fadeout to each trimmed sounds using SoX. This was done to remove any noise artifacts produced by trimming the sound. We didn't want our network to use this as a feature upon which to classify.

Each of the provided .wav files was then converted into a Numpy array so that it could be fed into the network. Each file had a sampling rate of 48 kHz. Thus, because each sound was trimmed to a length of 10 seconds, each Numpy array had 480,000 entries each corresponding to a timestep. In addition, the provided sound files were stereo audio. Thus, each entry had two channels of 480,000 entries each, meaning that each data point had a

dimension of 960,000.

Though this dataset provides us with the necessary ingredients to begin exploration of the aesthetic questions of interest, the small size of the dataset, as well as the skewed nature of the class distribution within the dataset, contributed to several difficulties in the design of our system.

First, the size of the dataset presents a problem for our goal. Deep learning typically requires large amounts of data in order to learn successfully. This becomes more pressing as the size of the network, and thus the number of the network's parameters, increases. While close to 1000 data points may seem like a lot of samples on which to train, this is actually a very small data set as compared to others used in deep learning projects. For example, the landmark effort in image classification on the ImageNet dataset by Krishevzky et al. [15] was trained on 1.2 million data points. This is a typical scale of a dataset on which deep learning networks are trained.

One of the main problems which a small dataset can cause is that of overfitting. When a large model is trained on a small dataset, one of the worst things that it can do is "memorize" the dataset. Instead of learning features that generalize across the classes which the network is being trained to classify, it can instead learn specific mappings for each data point in the training set. This can be especially difficult when the training data is of high dimension, as is this dataset. Often referred to as the "curse of dimensionality," the performance of a classifier typically increases as the number of dimensions in the data increase, to a point. After this, performance then often decreases as the dimensionality of the dataset keeps increasing [22]. This is because, as the dimensionality of the dataset increases, more data points are needed to adequately cover the possible space of data points. A small dataset will only cover a small portion of the possible space in which data points can lie, so learning features that generalize across the entire space can be tricky. Thus, our small dataset of high dimensionality makes

learning from it difficult.

There is an additional feature of our dataset which also contributes to difficulty in learning: the skewed nature of the class distribution. Of the 998 files supplied, only 164 were classified as "interesting." Thus, the class distribution of our dataset was 16.4% "interesting" and 83.6% "not interesting." As a note, we will often refer to the "interesting" class as the "positive" class and the "not interesting" class as the "negative" class.

It is often more difficult for a network to learn from a dataset that is skewed in class distribution. In the extreme case, a network can learn to predict merely the class that makes up the majority of the dataset. If the dataset distribution is skewed strongly enough, this can lead to excellent performance from the model, even though this is terrible from a standpoint of generalizability. In less drastic cases, this imbalance still leads to more difficulty in training. In each epoch of training, the network is seeing more data samples from the majority class in the dataset. Thus, it has more chances to adjust its parameters in such a way as to better classify these samples. It has fewer chances to adjust the parameters in order to classify the samples from the minority class, so these adjustments have the potential to be less nuanced. This in turn can lead to problems with the model overfitting the dataset.

These problems of training with skewed datasets are exacerbated when combined with a small dataset. Not only is the proportion of the minority class in the dataset small, but it is a small proportion of a small dataset. Thus, the number of data points from the minority class on which the network is trained is very limited. As previously described, this can lead to issues with the model learned, especially in terms of overfitting.

While unfortunate for training a deep neural network, both of these aspects of our dataset (small size and skewed class distribution) are inherent to the task at hand. It is very time consuming for Dr. Lyon to assign aesthetic labels to the sounds that Mushroom creates.

In discussion, he has mentioned the need to be essentially in the right "frame of mind" to perform this classification. Much as a composer cannot be expected to compose a musical score on command, a sound designer cannot necessarily assign an aesthetic judgment instantaneously; it is a creative process that requires time. Thus, any dataset of this nature will be inherently small. With the accumulation of judgments over time, it is certainly conceivable that a similar dataset could grow in size. It is, however, difficult to imagine this size approaching that of benchmark datasets used in tasks such as image recognition today. Image recognition datasets (e.g., ImageNet) require labeling of the image into pre-specified categories (e.g., beaver, school bus, burrito, etc.) Obtaining ground truth labels for classification tasks like these can be readily accomplished through crowd-sourcing, such as using Amazon Mechanical Turk. If we wish to obtain a network that represents the aesthetic preferences of a given individual (e.g., Dr. Lyon), then we have to have one person label all samples in our dataset. The fact that this type of aesthetic labeling is difficult to perform, and that the labeling task cannot be distributed, seems to suggest that any dataset obtained will be of limited size.

The fact that the class distribution of the dataset is skewed also makes sense. Mushroom is an oracular processor, so the sounds it generates are the result of random chains of processes being performed. It stands to reason that the majority of these sounds might be of no aesthetic interest to the user. This is backed up by the evidence presented by our dataset; only 16.4% of the 998 sounds were determined by Dr. Lyon to be interesting. Though this dataset is small, it does consist of almost 1000 different outputs of Mushroom run on a single source sound. These outputs are also spread among many different levels of processing. This suggests that the dataset is a typical distribution of the sounds available from Mushroom. Thus, it seems reasonable that the proportion of sounds found interesting will remain constant even as the size of the overall dataset grows.

If a small, class-imbalanced dataset seems to be inherent to the task of binary aesthetic classification, then something clearly needs to be done in order to mitigate these undesirable characteristics. Two ideas were devised in order to combat these problems: oversampling and windowing. Both of these will be discussed in the following section, but I will first discuss why a common technique known as data augmentation is not of use in this circumstance.

In order to increase the size of the dataset presented to a network for training, as well as increase the generalization capability of the network, a technique known as data augmentation is typically applied. Let's explain this in the context of an image classification task. Often, multiple augmented copies of images present in the dataset will be added. These augmentations take the form of transformations applied to the image originally contained in the dataset. These augmented images receive the label of the class to which the original image belonged. Thus, transformations are chosen under which classification should be invariant; the augmented image should be interpreted by the network in the same manner as the original, unmodified image. This means that common transformations include operations such as rotations and mirrorings. An image classification network should label an image as "CAT", even if the cat is upside-down or facing to the left instead of the right. Another common operation performed in data augmentation is blurring; a network should recognize a cat as a "CAT" even as the quality of the image in question decreases.

Unfortunately, these data augmentation ideas are hard to apply to our task. After all, we are unsure of the features which are necessary to perform aesthetic classification. This means that we are also unsure of the transformations under which classification should remain invariant. Perhaps "blurring" in a sound (or lack thereof) has a big impact on the aesthetic judgment of a sound. The temporal nature of audio data also rules out common operations performed on image data; it makes no sense for our classification to be invariant under a "mirroring" operation. As can be seen, the role of data augmentation in this task needs

more careful consideration. In the future work section of our conclusion, we detail a few ideas for exploration into meaningful data augmentation operations for classification of acoustic aesthetic judgment.

With data augmentation ruled out, we turn to other methods to alleviate the undesirable characteristics of our dataset in an effort to train our neural network. As mentioned, two ideas for this were devised: oversampling and windowing. First, we will describe what we mean by windowing.

The technique we call windowing gained inspiration from one of the previously discussed papers: [16]. In this work, they split the sounds in their dataset in segments with a length of 1-4 seconds. The network is trained to classify these truncated segments; when the trained network is asked to classify a full-length sound, it performs classification on truncated segments of the sound and then averages these predictions to classify the entire sound. In the original paper, this method is proposed as a means of reducing computational burden. In our case, we hoped to use this as a means of increasing the size of our dataset. We do this by making the assumption that the aesthetic labeling of a sound is relatively consistent throughout. In other words, if Dr. Lyon labeled a sound as aesthetically "interesting," he would also label truncated segments of that same sound as aesthetically "interesting." This assumption is more possible as the lengths of the truncated segments are closer to the full length of the sounds in the dataset. Thus, we are presented with a trade-off: we increase our dataset by more samples as we make the length of the truncated segments shorter, but the assumption that these segments mimic the aesthetic qualities of the untruncated sound grows weaker.

One problem introduced by windowing is discussed in the review of [16]: by performing classification on shorter segments of audio, we are losing out on larger scale features necessary for aesthetic evaluation. With the hierarchical representations of a CNN, helped by the

addition of maximum pooling (max pool) layers, our network has the capability of learning features that extend over long portions of the input sound. The scale of these features is relatively reduced when we perform classification on segmented inputs. Because we are not certain of the scales on which aesthetically relevant features lie, windowing is perhaps removing the possibility of finding them.

Another problem that is introduced by this windowing technique is also discussed in the review of [16]: the splitting of important features. Briefly, we wish to avoid splitting a salient feature of audio between two segments so as to go undetected. We reduce this possibility by overlapping the segments into which the original audio is split. The end of one segment of audio contains the beginning of another segment of audio. This provides a buffer preventing the separation of salient portions of audio, but the technique still remains vulnerable to the biases in scale of features introduced.

The other technique proposed to make the dataset more tenable for training a deep network was oversampling [17]. The idea is simple, though we have to be careful about considering the assumptions it begins to introduce. To augment a dataset that is skewed in its class distribution, additional copies of the underrepresented class can be added to the dataset. In our approach, these additional copies are randomly sampled from the samples of the minority class contained in the dataset. The goal is to increase the importance of learning features present in the underrepresented class. With very few samples in a dataset belonging to a class, it is not beneficial for the model to learn to distinguish this minority class within the dataset. It is able to achieve lower loss more quickly by beginning to learn how to accurately identify members of the majority class. This means that it is harder for the network to predict it is being given a sample from the minority class. When this class maps to an input being aesthetically "interesting," it seems that an inability to detect it is undesirable.

One parameter determining oversampling is the amount of copies we will add to the dataset.

Because the underlying goal of this operation is to balance the distribution of class labels present in the dataset, it makes sense to oversample to a point where the proportion of each class in the dataset is equal. As we will discuss in the Results section, we tried multiple variations on the number of data points oversampled. We found that, indeed, adding a number of copies such that the class proportions were equal resulted in the best results of the trained model.

Just as windowing biases the types of features learnable by the model, so does the technique of oversampling. By adding copies of the minority class, we are increasing the importance of the features contained in that particular sample to its class. This is in fact what increases the performance of our model. It does, however, begin to bias the distribution of the minority class that the model learns. The learned features present in the minority class samples have the potential to take on more importance than they actually possess in the "true" distribution of aesthetically interesting sounds. This could lead to issues of generalizability when introduced to new "interesting" sounds, sounds that were not present in the original (and thus the oversampled) dataset.

This issue grows weaker if we assume that the breadth of "interesting" sounds is adequately captured by the "interesting" sounds contained in the dataset. This is, unfortunately, exactly what is in question in the first place. We don't know what the aesthetically salient features of audio are, so we certainly can't determine if a sample of data adequately represents their possible space. Based on the relatively small number of sounds judged as "interesting," this is perhaps a safe assumption about our dataset. The overall small size of the dataset, however, does warrant caution. As with many methods discussed, there are both benefits gained and biases introduced. It becomes a matter of weighing the two to gain a model that it is both predictively successful and free of biases that obscure the truly relevant features for classification.

Chapter 5

Architecture

The main inspiration for the starting point of our network architecture comes from [21], whose architecture is presented in Figure 5.1. This being said, the network proposed within that paper is of a pretty standard organization for a CNN. Consisting of alternating convolutional layers and max pool layers, the network ends in a fully-connected (FC) layer which leads into a prediction later consisting of a softmax activation function.

Some architectural differences between the network of [21] and our network have already been discussed (e.g., the inclusion of Batchnorm). Another major difference between the two is the size: [21] has far more filters per layer than our network. The first three convolutional layers of ADD have 192 filters each, with the two deeper convolutional layers having 256 and 512 filters, respectively. Based on the much discussed limitations of our dataset, we are not able to support training a network of this size. It is actually difficult to assess the relative sizes of the dataset and the parameter set describing the network, as [21] does not provide the number of data samples on which the model was trained and validated. It is possible that [21] is utilizing the Million Song Dataset. This is the dataset used by [23], a paper referenced in the abstract of [21] as a work of which they "have followed in the footsteps." Additionally, this dataset lends itself to the task of learning track embeddings akin to those of collaborative filtering, as discussed in the review of [23]. So, a reasonable assumption of the size of the dataset used to train [21] might be that it was 1,000 times the size of our dataset. While it is probably safe to assume that their dataset is much larger than ours, its

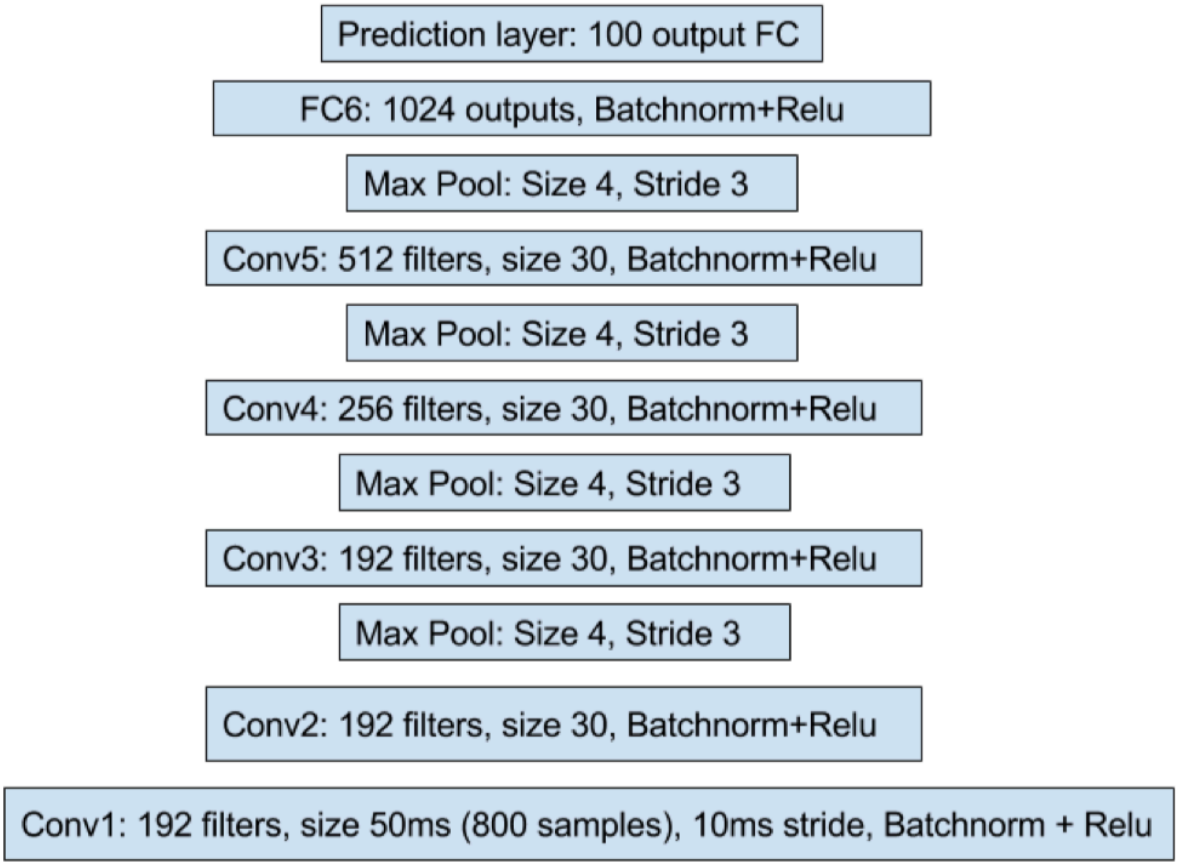

Figure 5.1: Audio DeepDream network architecture [21]

actual size is still unclear.

With a reduction in parameters necessary to reduce overfitting, two large options emerge: reduce the number of layers present in the network, or reduce the number of filters contained in each layer of the network. While ultimately reductions in both areas were made, the majority of reductions occurred in the number of filters contained per layer. This is due to our desire to preserve the hierarchical nature of the features learned by a CNN. As the visualizations produced by [19] seem to reveal, this organization seems to assist in the ability of classification. It also begins to provide a visual taxonomy of the features that the network learns. This is helpful when trying to interpret what are fundamental features in image recognition (e.g., lines and edges) and what are abstractions built from them (e.g., eyes and wheels). It seems like a reasonable idea to construct our network in this manner as well. After all, sound has many features across different timescales. This type of organization would be handled well by a CNN with several successive convolutional layers. Additionally, if this is indeed how aesthetically salient features are organized, we can begin to form a taxonomy of these features just as we can in the case of features learned for image recognition tasks.

Two other important features of [21] adopted by our network are strided convolutions and overlapping max pooling layers. In the first convolutional layer of [21], the authors implement what is known as strided convolution. This is a layer that, when convolving a filter across its input, moves the filter more than one step between each operation. This reduces the dimension of the output of the convolutional layer, as there are fewer overall convolutional operations performed. The authors propose using this in the first layer of their model, as "starting with raw audio means that there needs to be significant downscaling." As our raw audio input is also extremely high-dimensional, we too need to perform downscaling. This need is made more critical by the relatively small size of our dataset. Thus, we also decided to implement strided convolution within our first layer.

For the max pooling layers in [21], the authors chose to use overlapping regions over which to pool. This, too, seems like a reasonable choice. Along the same lines of the discussion of the windowing technique, we want to avoid the (potentially small) possibility that features of relevance are split between two regions of consideration. In the worst case, a feature would be split in such a way that the amount contained in either region is not enough for the feature to be recognized in either region. Just as an overlapping approach to the windowing technique attempts to mitigate this possibility, so too does an overlapping approach to max pooling. There are far fewer possibilities for features being split between regions when this approach is taken. This benefit outweighs the slight increase in parameters necessary to implement this operation.

In the discussion of the original [19], we discussed the optimization approach taken to visualize the filters learned by a CNN. We will provide a brief refresher here, with a focus on the way that this was implemented in our system.

Instead of optimizing the weights of a network in reaction to the way they transform an input image, we instead optimize over the input image itself. We do this with the goal of maximizing the activation of a given filter or layer. Thus, after many consecutive iterations of this procedure, we obtain an image that maximally activates the portion of the network under consideration. In other words, we can interpret the image as having the type(s) of features that the chosen portion of the network has learned to detect. Of course, we are working with sound files instead of images, but the underlying process and motivation behind it remains the same.

As also discussed in the section on [21], many implementations of "dreaming" also involve forms of regularization to improve results. We mentioned several reasons why we avoid these specific ideas in our project, though this does not mean we are opposed to the overall idea. Careful thought into what features (on what scale) can be regularized needs to be conducted

first. When we perform regularization on the optimized images, we are effectively removing information from the image. In the case of the techniques proposed for image regularization, information being removed is irrelevant to the actual way we interpret images; for example, they can help prevent the visualizations being those of adversarial examples. They help constrain the outputs into the space in which real-life images reside. It is not immediately intuitive what form this would take in the audio domain. We discuss some potential areas of exploration for this question in the future work section of our conclusion.

Our optimization objective is to maximize the mean activation of a given filter, where the mean is taken over the outputs of that filter. We performed this optimization both before and after the ReLU activation was applied to the layer. In the Results section, we compare the "dreams" of the pre-activation and post-activation methods.

One additional note concerning the optimization process is the constraining of values. Because we normalize our inputs being fed into the network, the values that the network sees as it is training always reside between -1 and 1. This means that we don't want our "dreamed" sounds to contain values outside of this range. Thus, we implement a simple clipping procedure following each iteration of optimization. Any values that have been pushed outside of the [-1, 1] range by gradient ascent are reduced back to these boundary values. Not only does this help to keep the "dreamed" sounds closer to the actual sounds contained in the dataset, but it also becomes necessary when dealing with ReLU activation functions within the network. Because the output of a ReLU activation function is unbounded in the positive region, we can theoretically keep increasing these activations further and further. Thus, we want to constrain the regions of "dreaming" to those that are valid. This constraint also allows us to listen to the sounds "dreamed" by our algorithm. If the "dreamed" values lie outside the range of the normalized inputs, then the denormalized "dreamed" values will lie outside the range of the values of the raw inputs in our training set. This could potentially

cause problems when we write these to a .wav file so that we can hear the results of our algorithm. Thus, from both a practical and intuitive standpoint, we apply this constraint to our optimization procedure.

The final network design was informed by [21], though we had to trade some of the size (and corresponding representational capacity) to effectively learn from our tiny, class-skewed dataset. Our final network consisted of four convolutional layers, with the deeper three each followed by a max pooling layer. These layers contained 8, 16, and 32 filters, respectively. Following these layers was a fully connected layer of 32 nodes. Finally, the network ended with two nodes containing a softmax activation (allowing us to make a binary classification). This architecture can be seen in Figure 5.2.

As one might assume, the first convolutional layer is a bit of a bottleneck for our network. This is the layer that receives an incredibly high-dimensional feature vector as input, reducing the dimensionality substantially before it is passed onto the next convolutional layer. Another trade-off in the battle of overfitting occurs here. As we reduce the dimensionality of the layer's output, we require fewer parameters in our model. This is helpful in resisting the overfitting of our model, but it also reduces the information that the deeper layers of our network can access. The majority of dimensionality reduction in the network occurs in this top layer, so a reduction here in the amount of information output affects the network more substantially than a reduction in output size for any other layer. In turn, allowing more information to pass also greatly increases the number of parameters necessary in the model. Because the first layer is where the majority of reduction occurs in this network, it also means that this is where a majority of parameters exist.

Ultimately, we arrived on the solution of having the first convolutional layer consist of one filter. This filter is trainable, but its weights are initialized to values of 1. The rest of the weights in our network are initialized using the Xavier normal initializer [9]; this procedure

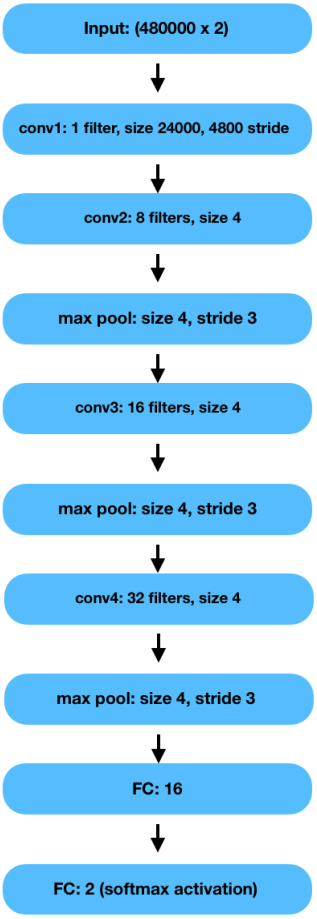

Figure 5.2: Network architecture

initializes the values of the weights as normal random variables. The motivation behind initially setting all of the first layer's weights to 1 is to essentially to avoid the introduction of noise. Because there is a huge number of parameters within this layer, a lot of them are not adjusted throughout training (especially on our small dataset). Thus, we basically have random noise as the weights for this layer upon training. By setting the weights to one, we are letting most of the input data pass through with any learned weights providing small adjustments, instead of "fuzzing" up the sound with random normal weights.

Chapter 6

Results

The confusion matrix for our network trained using the windowing approach on our dataset is contained in Table 6.1. We split each original 10 second audio file into segments of 5 seconds each, with an overlap of 4 seconds between adjacent segments. This means we trained on 4191 samples and validated on 1797 samples (70%-30% split of the entire 5988 windowed samples). This 70%-30% split for training and validation sets is used for all methods.

Table 6.1: Confusion matrix for network using the windowing approach.

	y=1	y=0
$\hat{y}$=1	0	0
$\hat{y}$=0	298	1499

The results from training our network on the original, 998 sample dataset (without performing oversampling) are contained in Tables 6.2 and 6.3.

Table 6.2: Confusion matrix for network without oversampling.

	y=1	y=0
$\hat{y}$=1	40	14
$\hat{y}$=0	14	232

The results when using a dataset oversampled with five times the number of samples originally in the minority class (for a total of 1654 samples within the dataset) are presented in Tables 6.4 and 6.5.

Table 6.3: Performance metrics for network without oversampling.

Precision	.741
Recall	.741
Specificity	.943
Negative Predictive Value	.943

Table 6.4: Confusion matrix for 5x oversampling.

	y=1	y=0
$\hat{y}$=1	239	32
$\hat{y}$=0	13	213

Some of the visualizations resulting from applying our "dreaming" algorithm to our trained network are contained in Figures 6.2 through 6.14 below. We have chosen to include some interesting visualizations in the body of this work, with many additional filter visualizations contained in Appendix.

We chose to not start with an input of white noise as in [19]. Rather, we instead used as input the original source sound fed into Mushroom to create our dataset. It consists of a trombone being played, with the waveform displayed in Figure 6.1.

The layers and filters shown in these visualizations are chosen using a few criteria. First, there are many extremely similar filters learned within a given layer. Thus, we have chosen to show filters that represent the diversity of those learned. Second, the network also seems to learn what [14] terms "dead" filters. These are filters that don't appear to be activated by the supplied input images. We first see this phenomena as we are performing the optimization

Table 6.5: Performance metrics for 5x oversampling.

Precision	.882
Recall	.948
Specificity	.869
Negative Predictive Value	.942

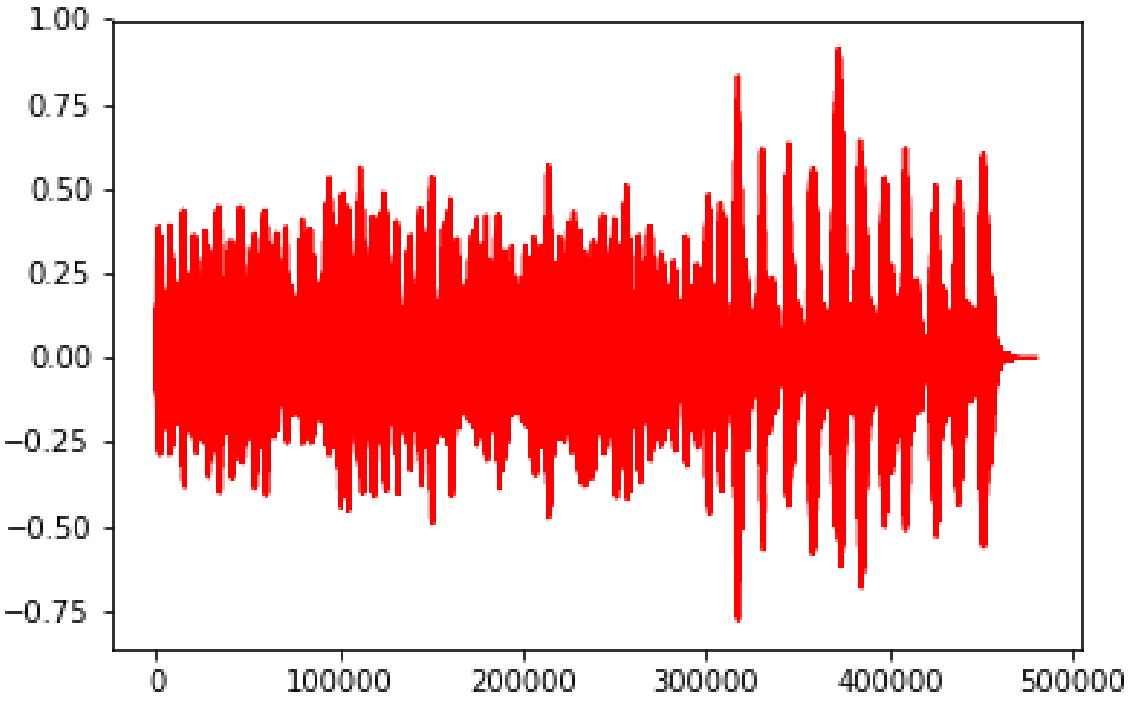

Figure 6.1: Initial sound as input for ”dreaming”

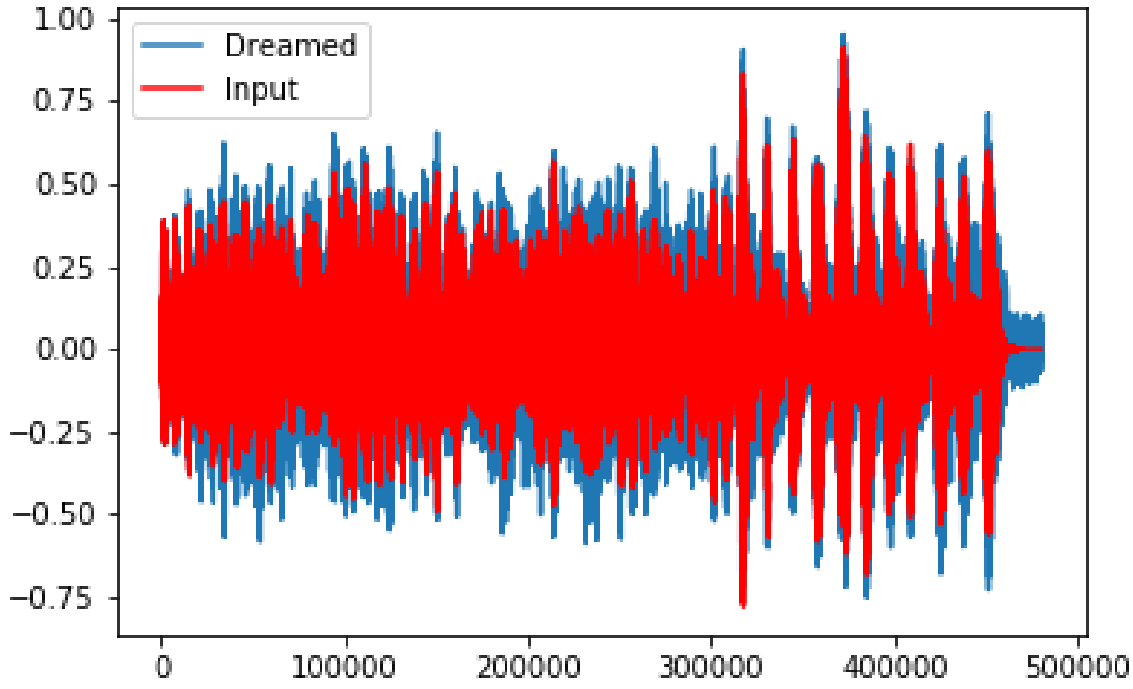

Figure 6.2: Visualization of layer 2, filter 1

procedure. After each iteration of optimization over the activation of one of these “dead” filters, the activation value of the filter induced by the input either remains at zero or very slowly increases. In the increasing case, the rate of change is far less than that of filters which do not appear to be dead. As these visualizations are not particularly revealing (or at least no more revealing than the mere fact that they exist), we have chosen to present filters that are not “dead.” Figure 6.15 contains the visualization of a ”dead” filter for reference.

One slight modification we can make in the visualization process is whether to maximize the activation before or after the layer’s ReLU activation function is applied. In fact, in the GitHub repository for Tensorflow’s DeepDream implementation [5], the authors optimize over the outputs of the layer before the ReLU activation is applied. They claim that this

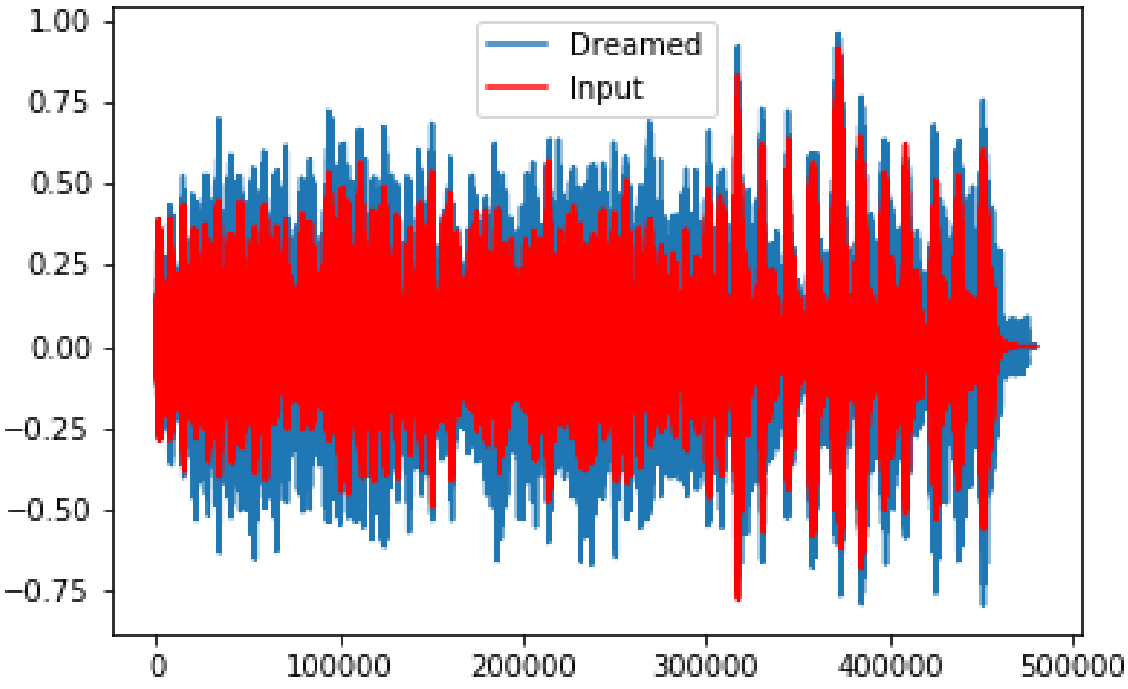

Figure 6.3: Visualization of layer 2, filter 3

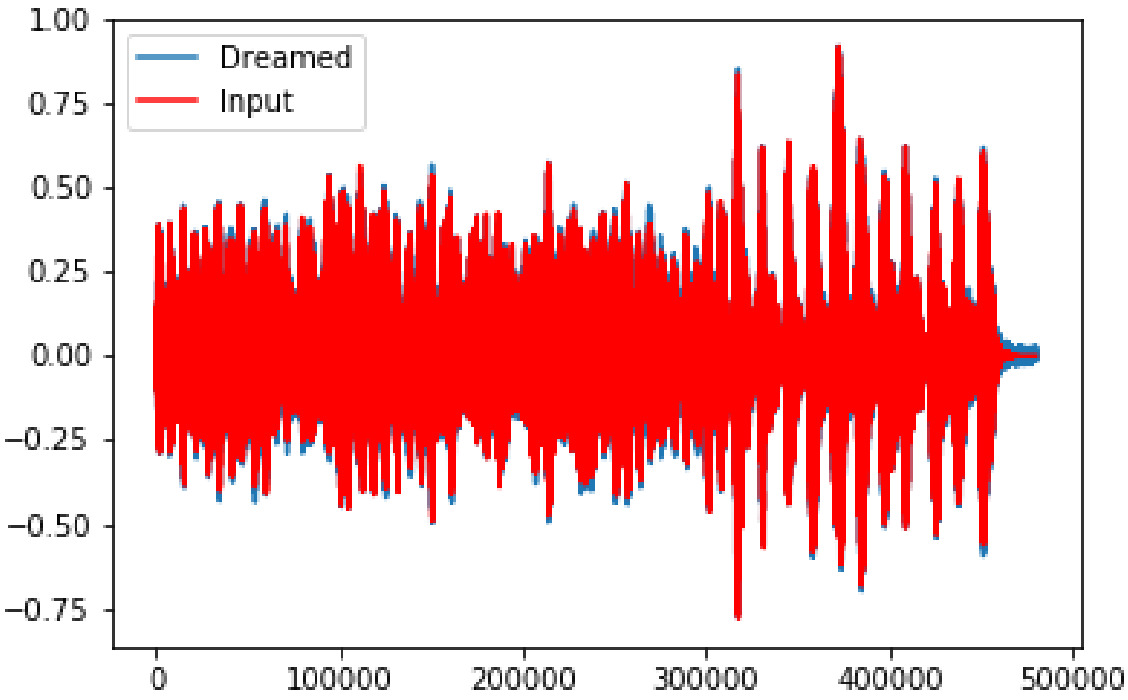

Figure 6.4: Visualization of layer 2, filter 7

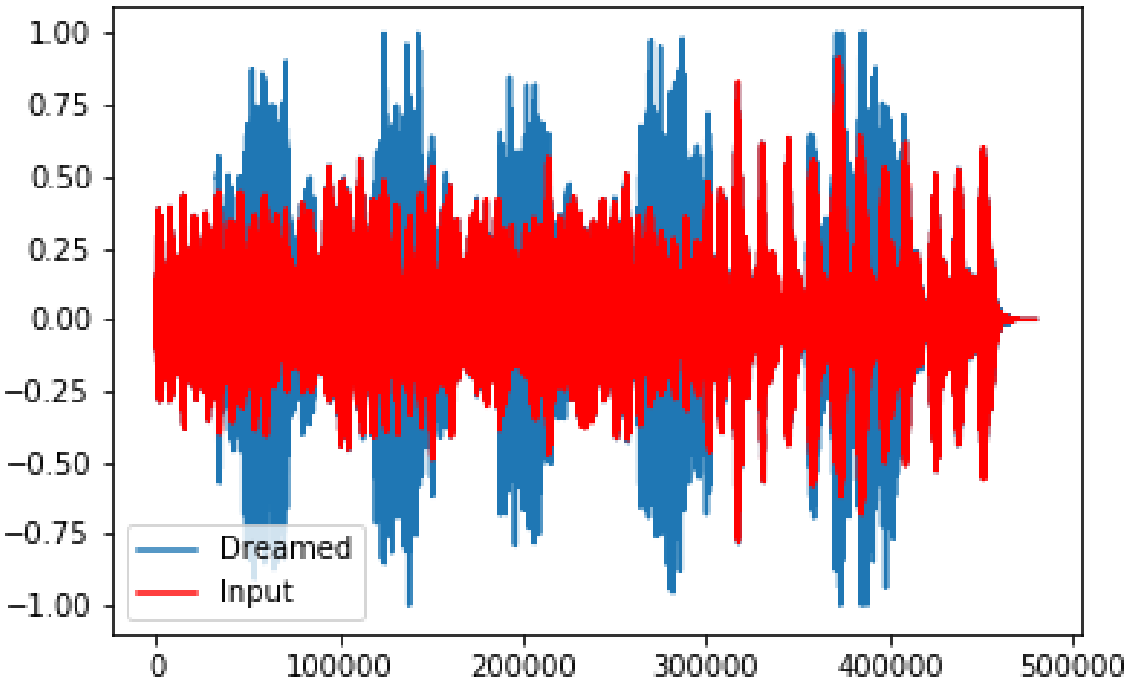

Figure 6.5: Visualization of layer 3, filter 2

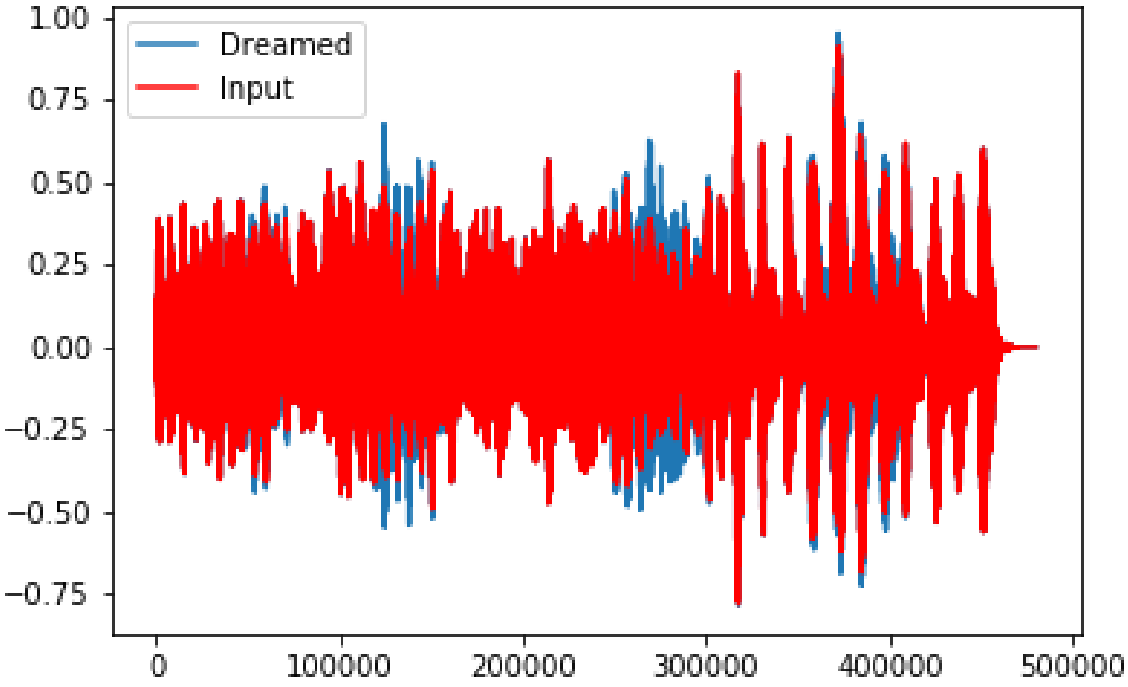

Figure 6.6: Visualization of layer 3, filter 7

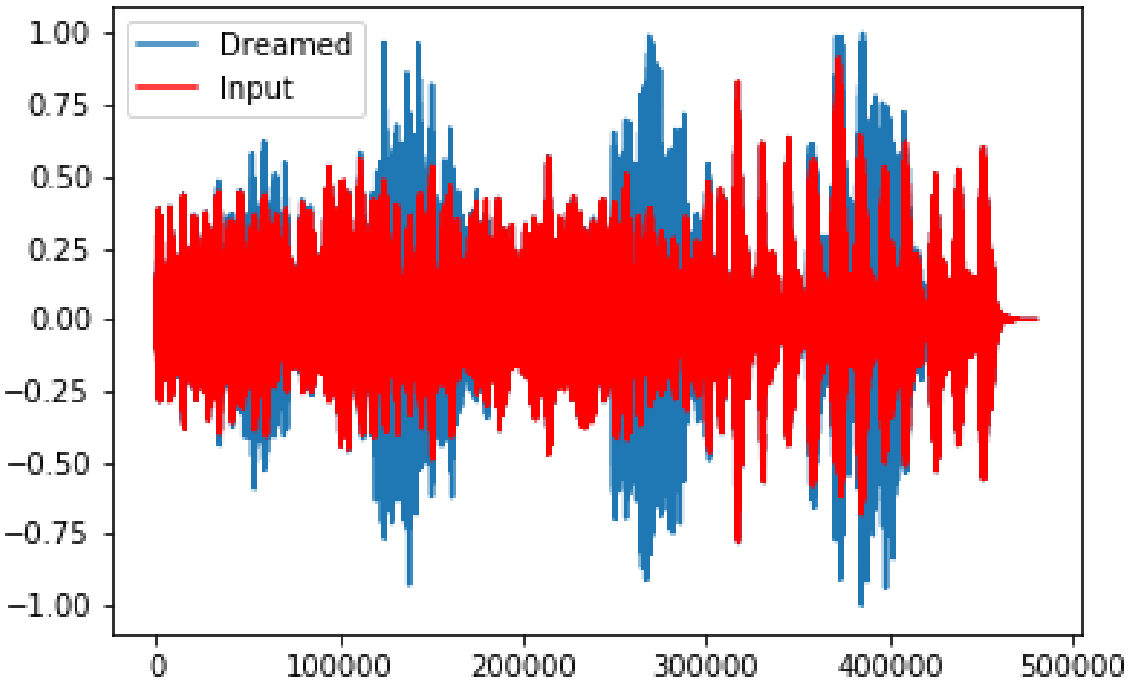

Figure 6.7: Visualization of layer 3, filter 11

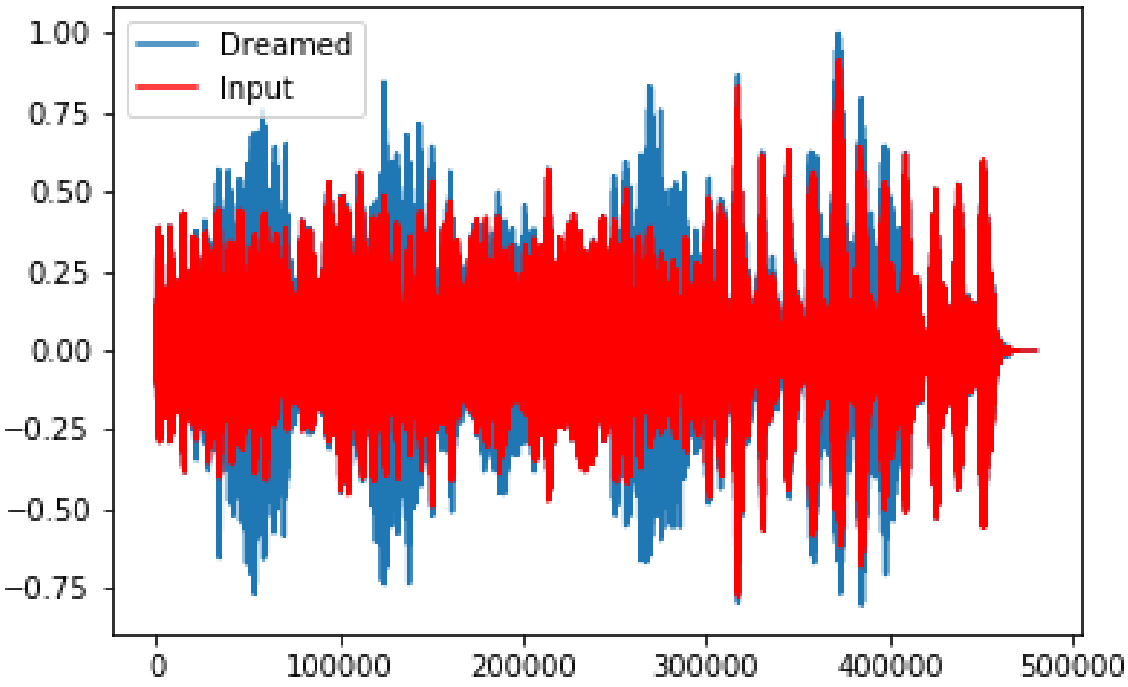

Figure 6.8: Visualization of layer 3, filter 15

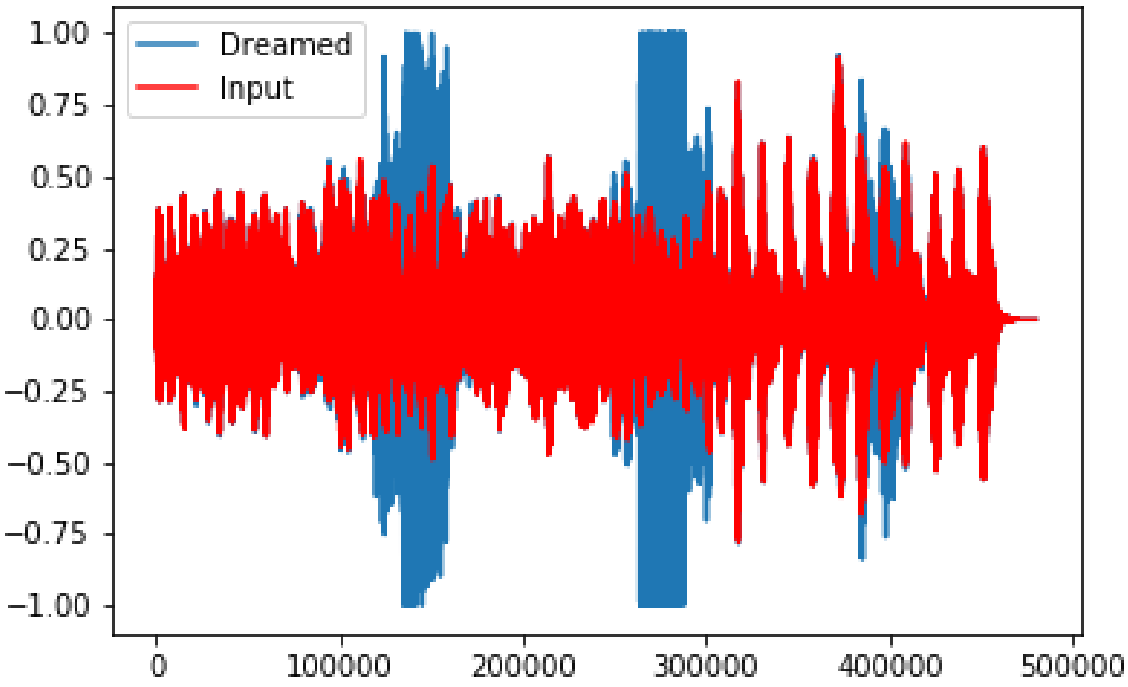

Figure 6.9: Visualization of layer 4, filter 4

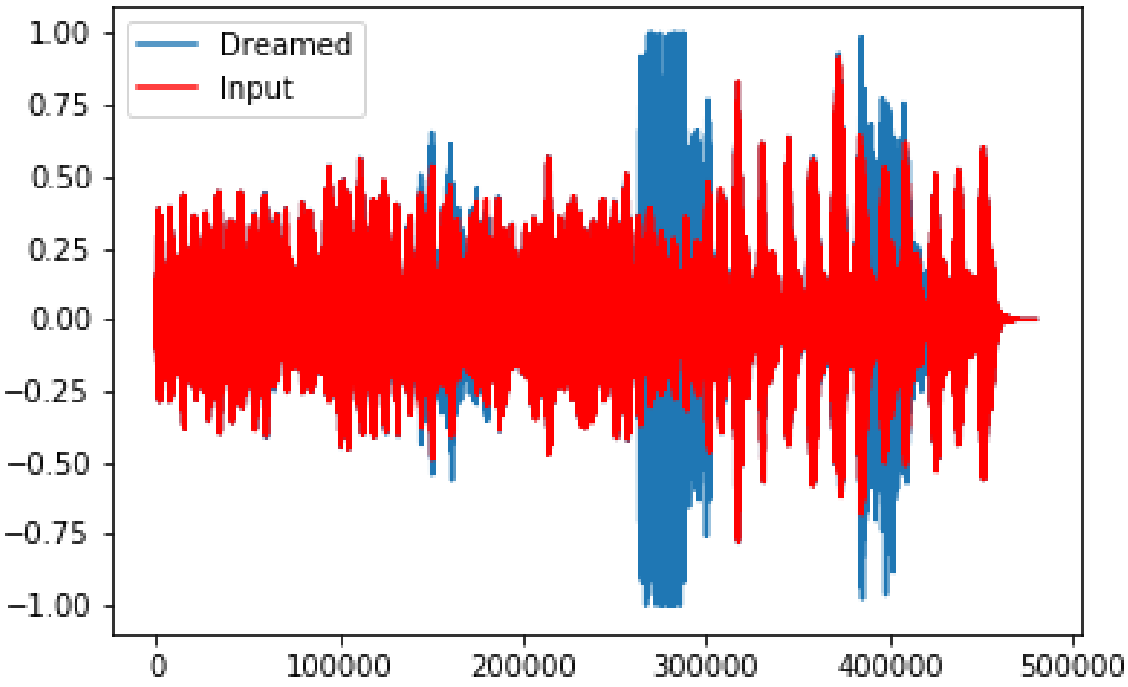

Figure 6.10: Visualization of layer 4, filter 10

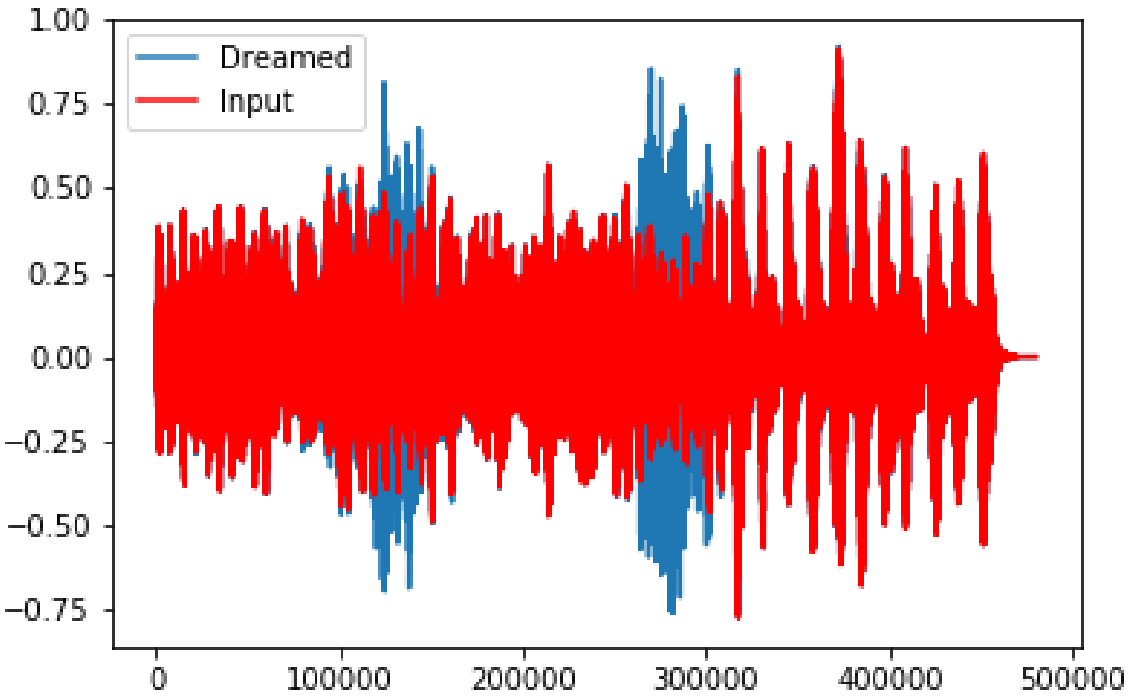

Figure 6.11: Visualization of layer 4, filter 15

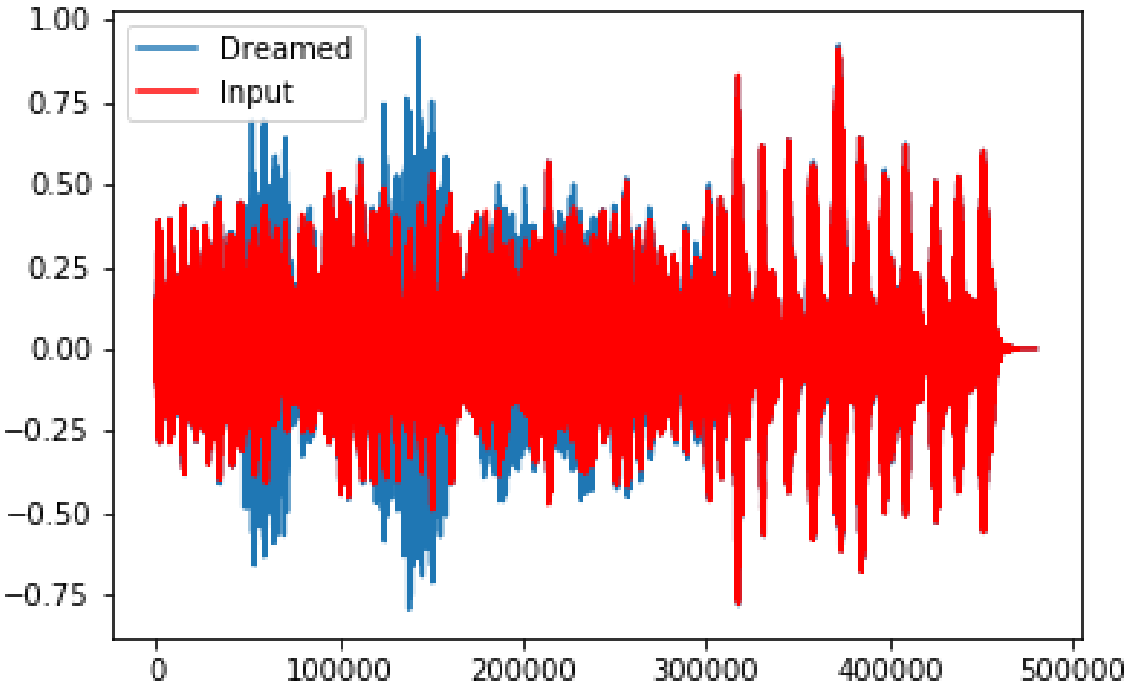

Figure 6.12: Visualization of layer 4, filter 19

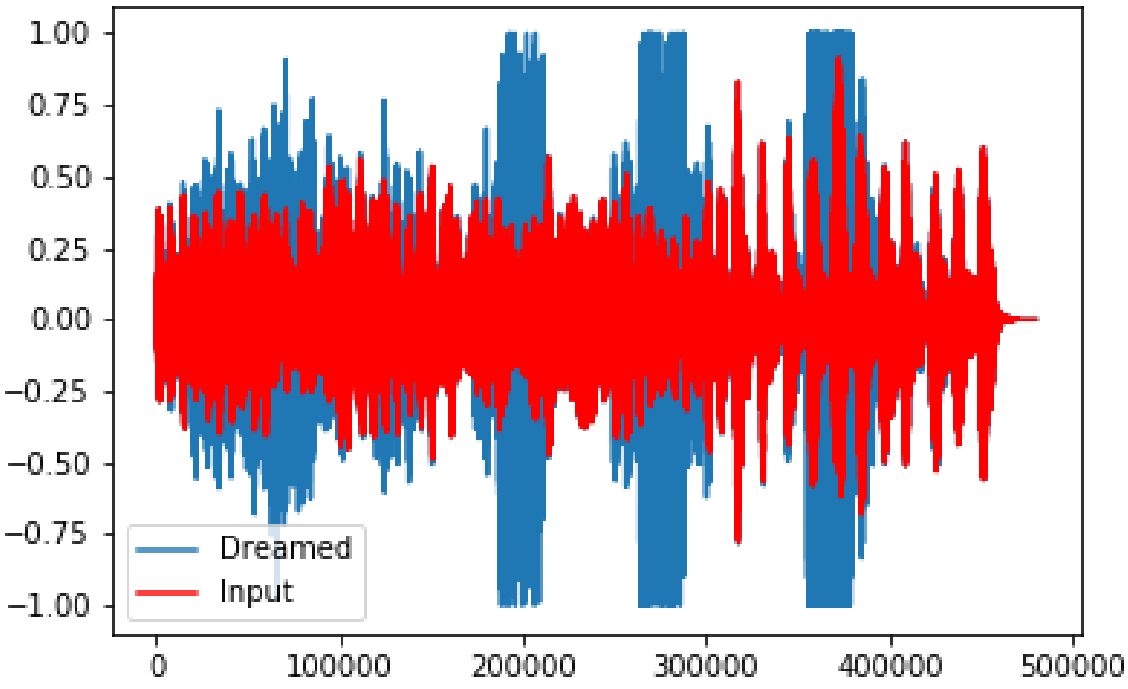

Figure 6.13: Visualization of layer 4, filter 21

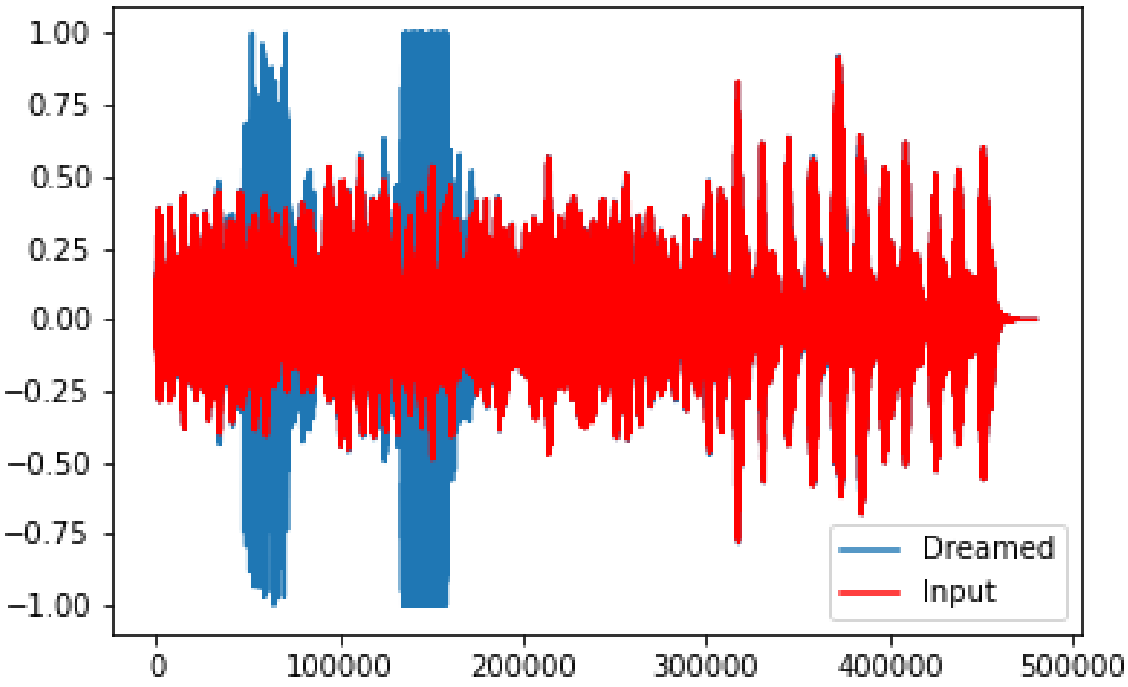

Figure 6.14: Visualization of layer 4, filter 30

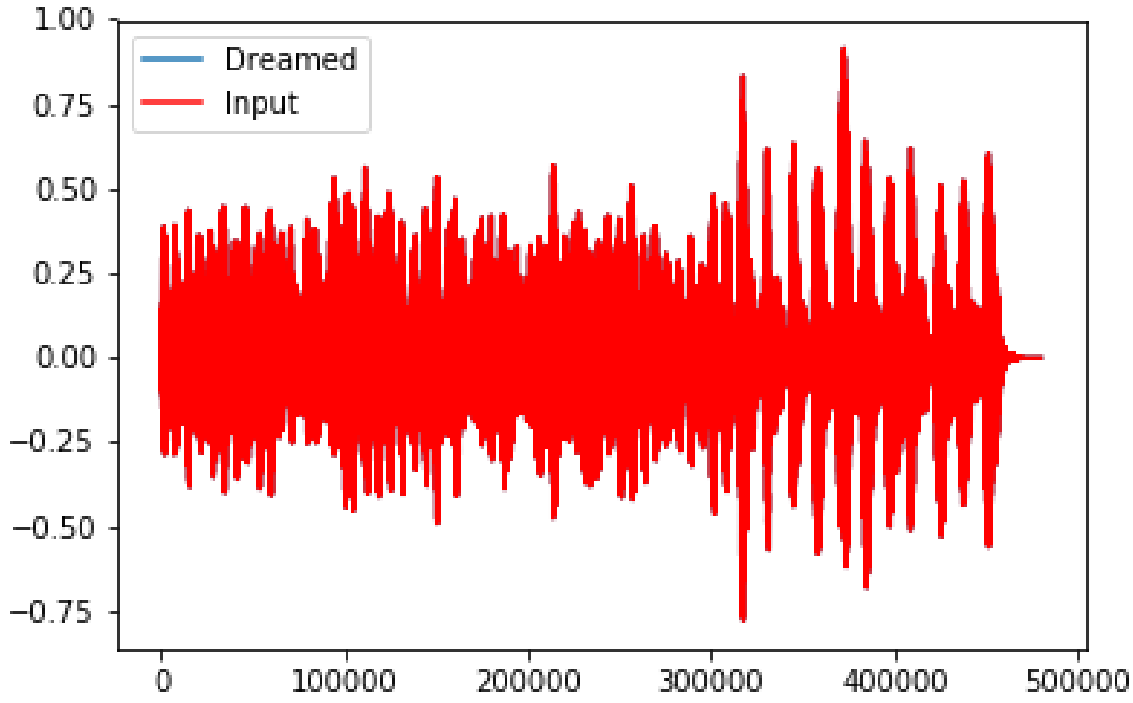

Figure 6.15: Visualization of ”dead” filter: layer 4, filter 17

allows them to “have non-zero gradients for features with negative initial activations.” In other words, a ReLU function will treat any negative input in the same way: output a value of zero. This makes it difficult to determine the impact that these negative activations are having on the overall performance. 6.16 contains a an example of a visualization created by “dreaming” over the filter activations before a ReLU nonlinearity was applied. As with the visualizations “dreamed” after the ReLU activation, we chose to visualize filters that did not appear “dead.”

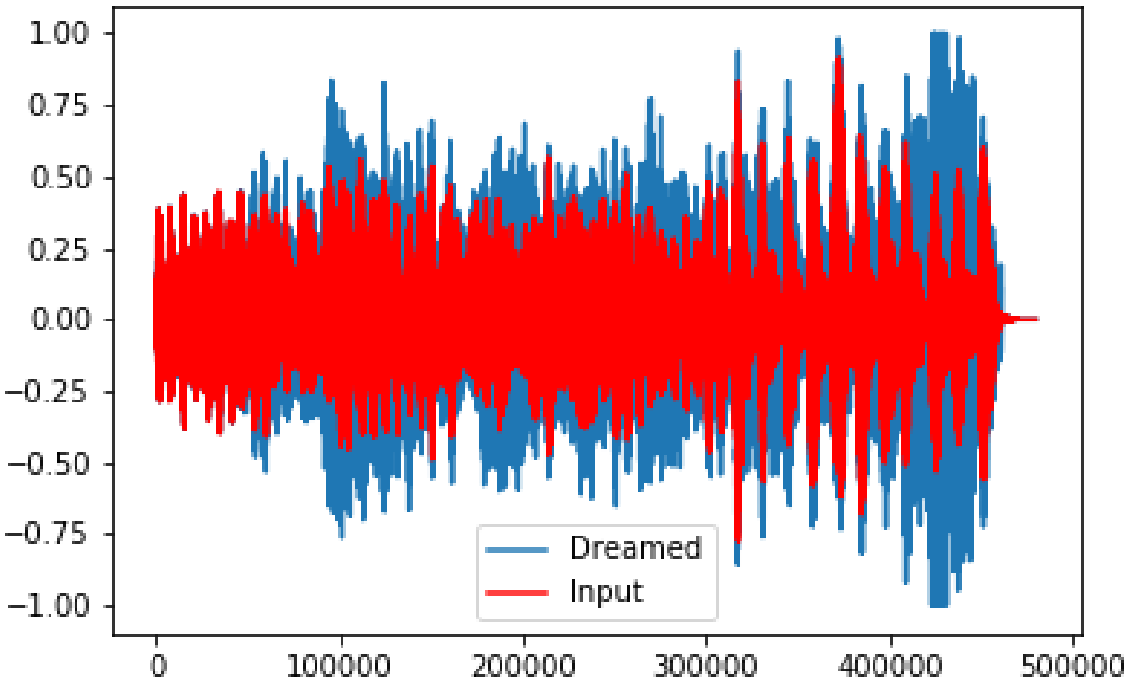

Figure 6.16: Visualization of filter before ReLU activation: layer 3, filter 1

Chapter 7

Conclusion

At the outset, this project attempted to do several things. First, we wanted to build a model with the capacity to classify sounds as aesthetically "interesting" in the same manner as a professional composer. In addition, we hoped to probe the representations learned by this model to help us interpret them. Using a "dreaming" algorithm on our network would allow us to both see (and thus hear) the features that our model learned, features that are helpful and fundamental in determining aesthetic relevance. We also worked to avoid (or at least mitigate) biases about the types of features useful for this task; we wanted to allow the network to determine the fundamental features for acoustic aesthetic classification.

Several features of the dataset, which we argue are inherent to the classification task at hand, caused issues in the model's learning of aesthetically salient features. Namely, the small size of the dataset and the skewed distribution of its classes make the training of a CNN difficult. Multiple methods of mitigating these issues were explored, with the eventual decision being to use the oversampling technique described. We chose oversampling as this causes (relatively) minimal damage to the features in which we are interested. Nevertheless, we should still be careful to note that the features we are learning do not map directly to the preferences of Dr. Lyon as originally hoped; rather, they are modified by the changes made to the dataset in an attempt to enable learnability for our model. With this qualification in mind, we were able to design and train a network that mimics the aesthetic classification of a professional composer (Dr. Eric Lyon). We were also able to begin to both visualize and

hear the types of features for which the network ”listens” through algorithmic ”dreaming.”

These visualizations represent an important first step in interpreting the fundamental aesthetic features of sound. Clear patterns can be observed in the “dreamed” outputs of our network, as well as heard when we listen to the outputs. Segments of audio are emphasized within these visualizations, allowing us to begin to determine important sections of audio for aesthetic classification. The successful predictive performance of our network lends further credence to these feature audiations.

Two areas of future exploration immediately come to mind, as mentioned in the preceding sections: data augmentation and visualization regularization. Both of these ideas are important to a system as described in this paper. The oversampling performed can be thought of as a form of data augmentation, though it is not as nuanced as techniques used for image datasets. Careful thought into meaningful augmentation operations for audio data would assist in the training of DNNs for this task. One rough thought would be to augment frequencies on a human-imperceptible scale. If we wish to uncover fundamental aesthetic features present in sound, we would like the aesthetic classification of our network to be invariant to features that are meaningless to a human performing this task.

Along these same lines, we should apply these invariances to the visualization (and thus audiation) of network features. We would like our “dreams” to possess characteristics of actual sound in the hopes of preventing the ”dreaming” of adversarial examples. As noted, this is a common practice in the image domain; it would be a useful exercise to consider analogous invariances for the audio domain.

Overall, this project represents the initial steps toward studying the determinants of acoustic aesthetic classification. By creating a network to mimic the judgments of a professional composer, while minimizing any biases of feature importance introduced, we have enabled

a unique opportunity to begin to understand and interpret these aesthetic components of sound. By discussing the difficulties that are encountered in a dataset for this task, as well as offering solutions to these problems that cause minimal damage to the features of interest, we hope to provide a foundation for further exploration into using deep neural networks to determine fundamental aesthetic qualities of sound.

Bibliography

[1] Mushroom. https://nmbx.newmusicusa.org/Mushroom/. Accessed: 2019-05-13.

[2] MagnaTagATune Dataset. http://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset. Accessed: 2019-05-13.

[3] Million Song Dataset. http://millionsongdataset.com. Accessed: 2019-05-13.

[4] SoX - Sound eXchange. http://sox.sourceforge.net/soxi.html. Accessed: 2019-05-13.

[5] DeepDreaming with Tensorflow. https://github.com/tensorflow/tensorflow/blob\/master/tensorflow/examples/tutorials/deepdream/deepdream.ipynb. Accessed: 2019-05-13.

[6] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848.

[7] Sander Dieleman, Philemon Brakel, and Benjamin Schrauwen. Audio-based music classification with a pretrained convolutional network. In *12th International Society for Music Information Retrieval Conference*, pages 669–674, October 2011.

[8] C Fellbaum. *WordNet: An Electronical Lexical Database*. Bradford Books, 1998.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings*

of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

[10] D. Griffin and Jae Lim. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, April 1984. ISSN 0096-3518. doi: 10.1109/TASSP.1984.1164317.

[11] Henkjan Honing. *The Origins of Musicality*. The MIT Press.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Pisa, Italy, December 2008. IEEE. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.22. URL http://ieeexplore.ieee.org/document/4781121/.

[13] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, February 2015. URL http://arxiv.org/abs/1502.03167. arXiv: 1502.03167.

[14] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. URL http://cs231n.github.io/understanding-cnn/.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey 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. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[16] Jongpil Lee, Jiyoung Park, Keunhyoung Luke Kim, and Juhan Nam. Sample-level Deep Convolutional Neural Networks for Music Auto-tagging Using Raw Waveforms. *arXiv:1703.01789 [cs]*, March 2017. URL http://arxiv.org/abs/1703.01789. arXiv: 1703.01789.

[17] Charles X. Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 73–79. AAAI Press, 1998. URL http://dl.acm.org/citation.cfm?id=3000292.3000304.

[18] Eric Lyon. Mushroom – An Oracular Sound Processor. URL http://disis.music.vt.edu/eric/LyonPapers/Mushroom_Oracular.html.

[19] Alexander Mordvintzev, Christopher Olah, and Mike Tyka. Inceptionism: Going Deeper into Neural Networks, June 2015. URL https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[20] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2(11), November 2017. ISSN 2476-0757. doi: 10.23915/distill.00007. URL https://distill.pub/2017/feature-visualization.

[21] Adam Roberts, Cinjon Resnick, Diego Ardila, and Doug Eck. Audio deepdream: Optimizing raw audio with convolutional networks, 2016. URL https://18798-presscdn-pagely.netdna-ssl.com/ismir2016/wp-content/uploads/sites/2294/2016/08/ardila-audio.pdf.

[22] G.V. Trunk. A Problem of Dimensionality: A Simple Example. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):306–307, July 1979.

[23] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based

music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf.

Appendix A

Filter Visualizations

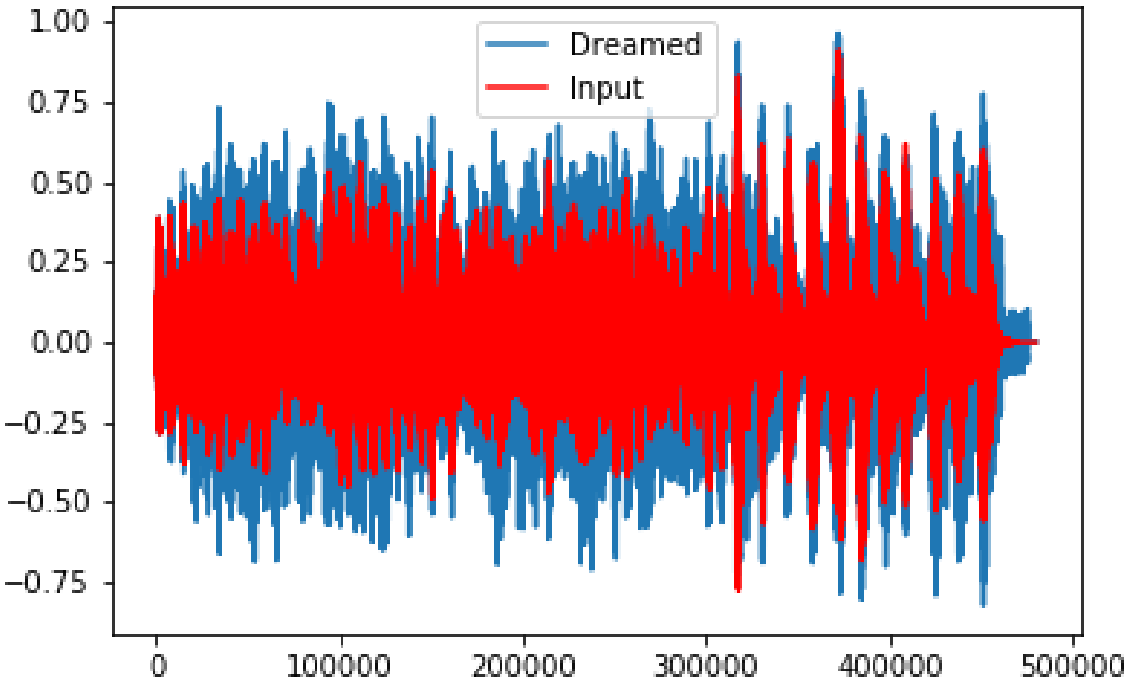

Figure A.1: Visualization of layer 2, filter 0

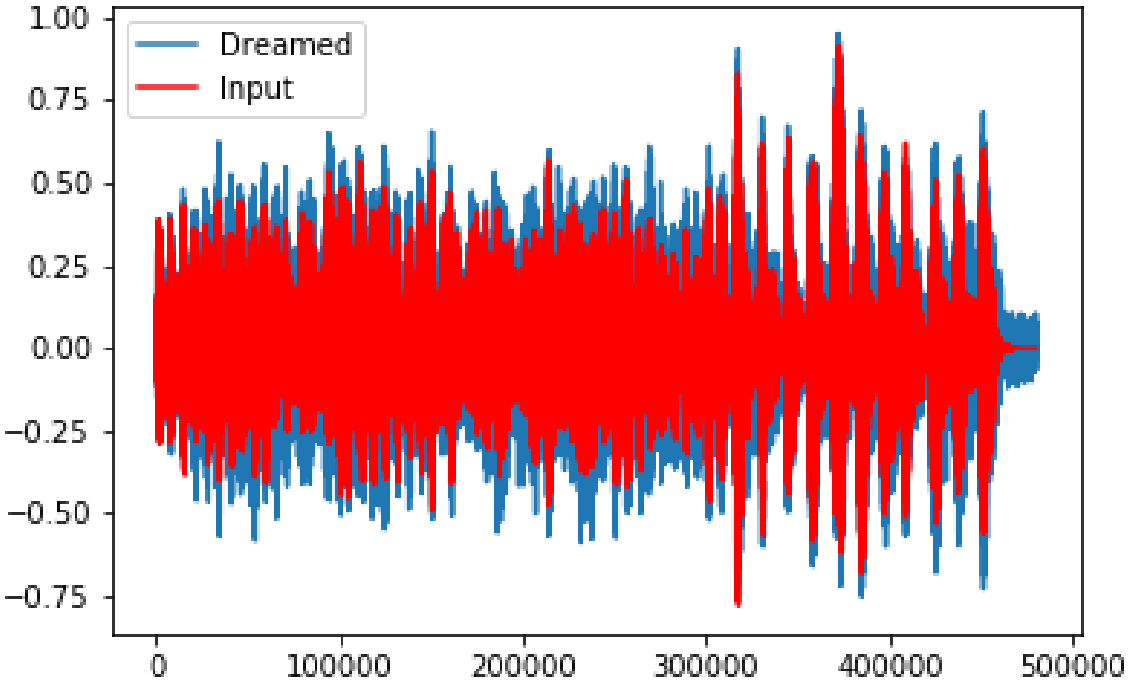

Figure A.2: Visualization of layer 2, filter 1

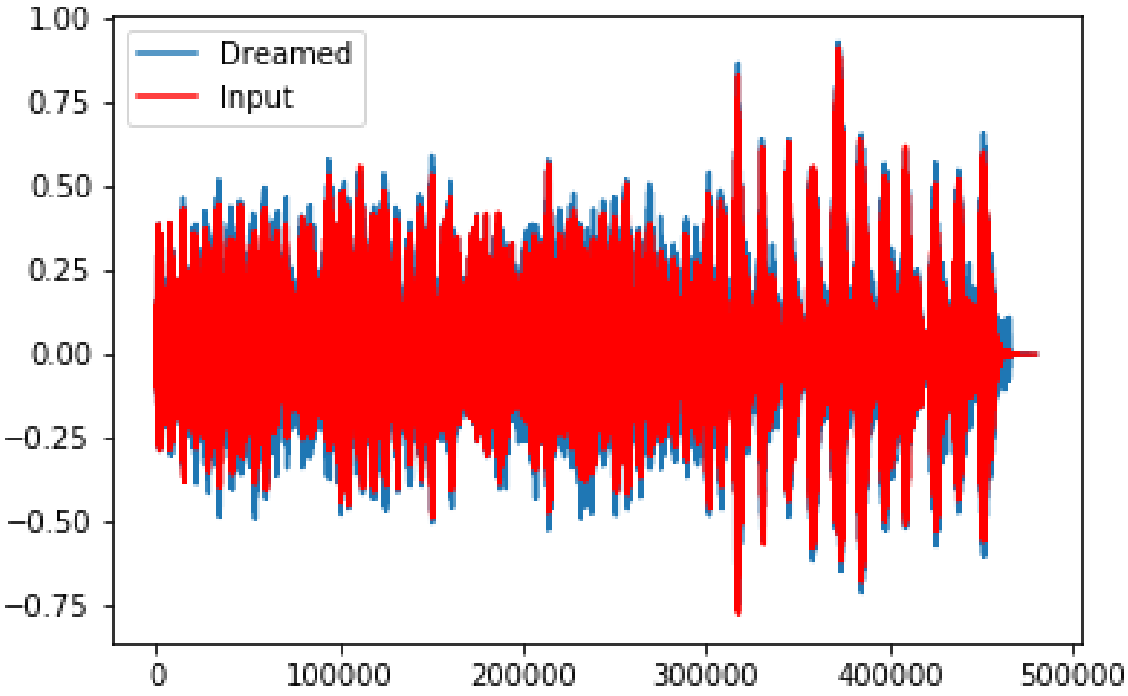

Figure A.3: Visualization of layer 2, filter 2

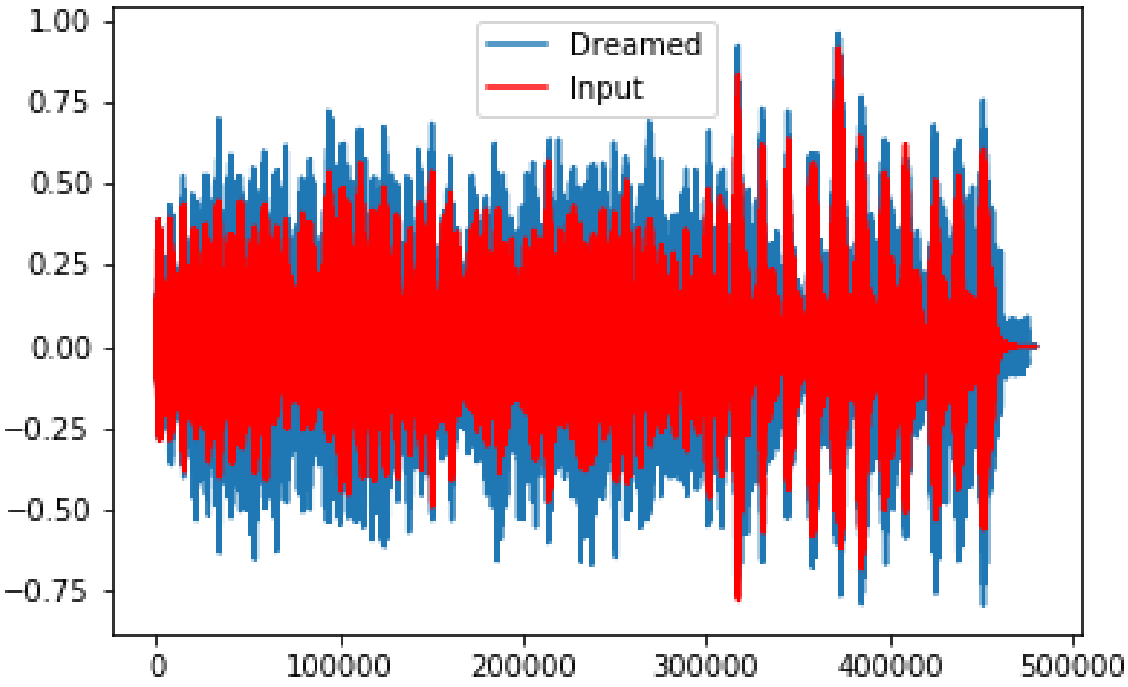

Figure A.4: Visualization of layer 2, filter 3

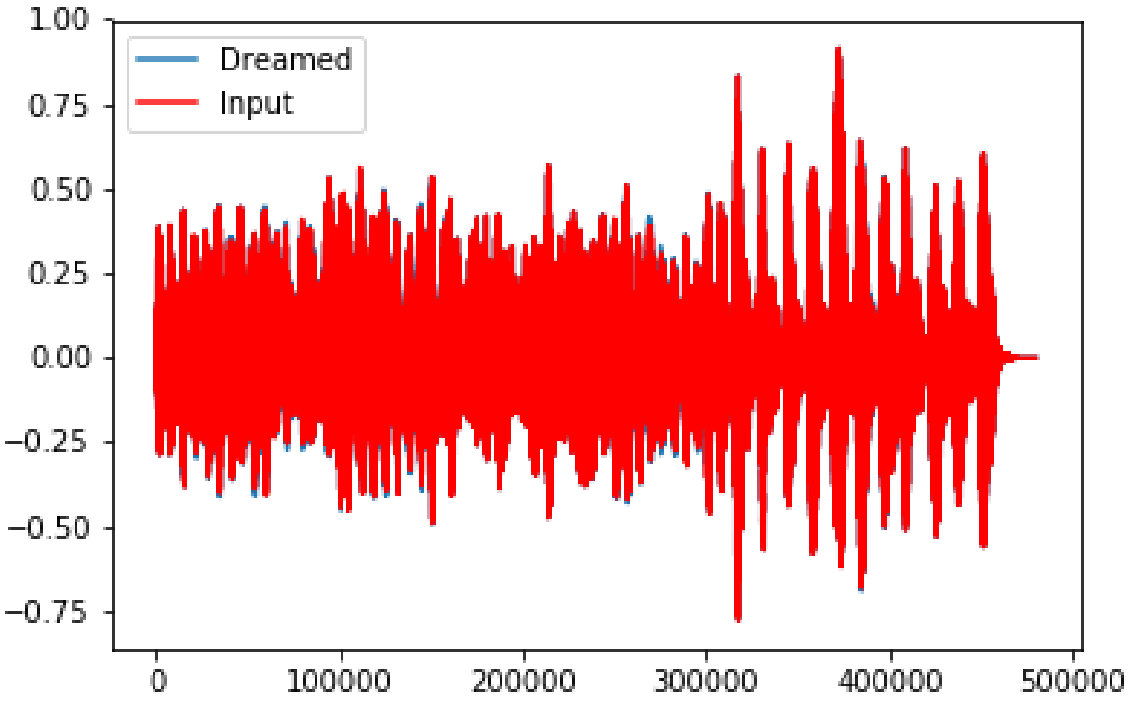

Figure A.5: Visualization of layer 2, filter 4

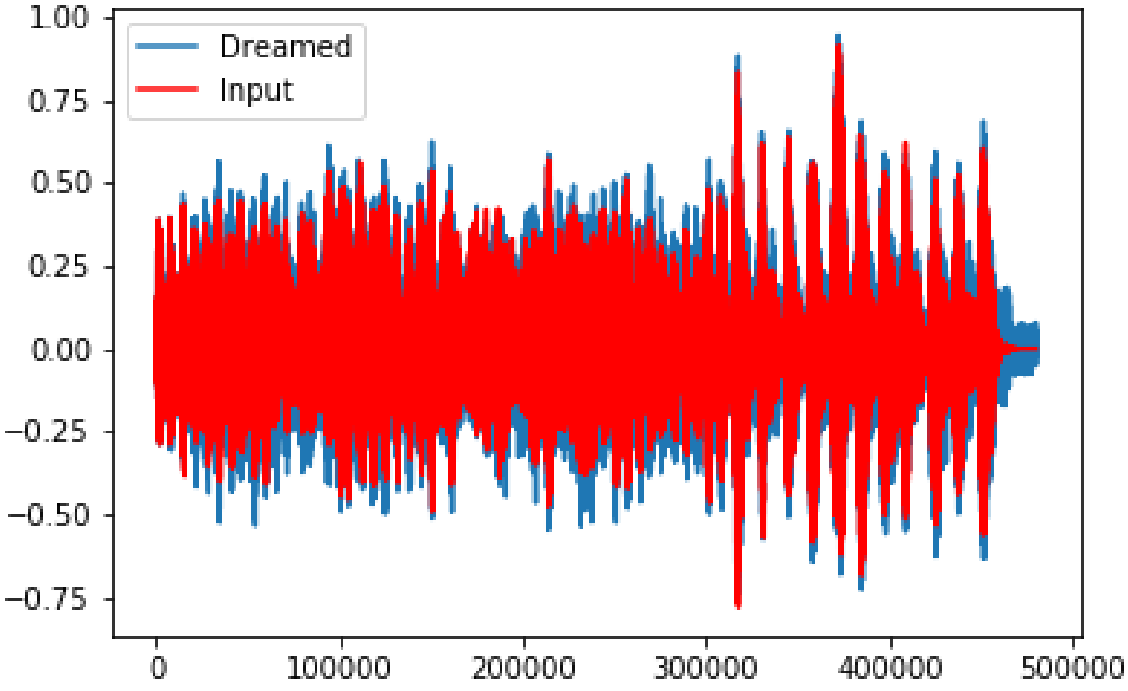

Figure A.6: Visualization of layer 2, filter 5

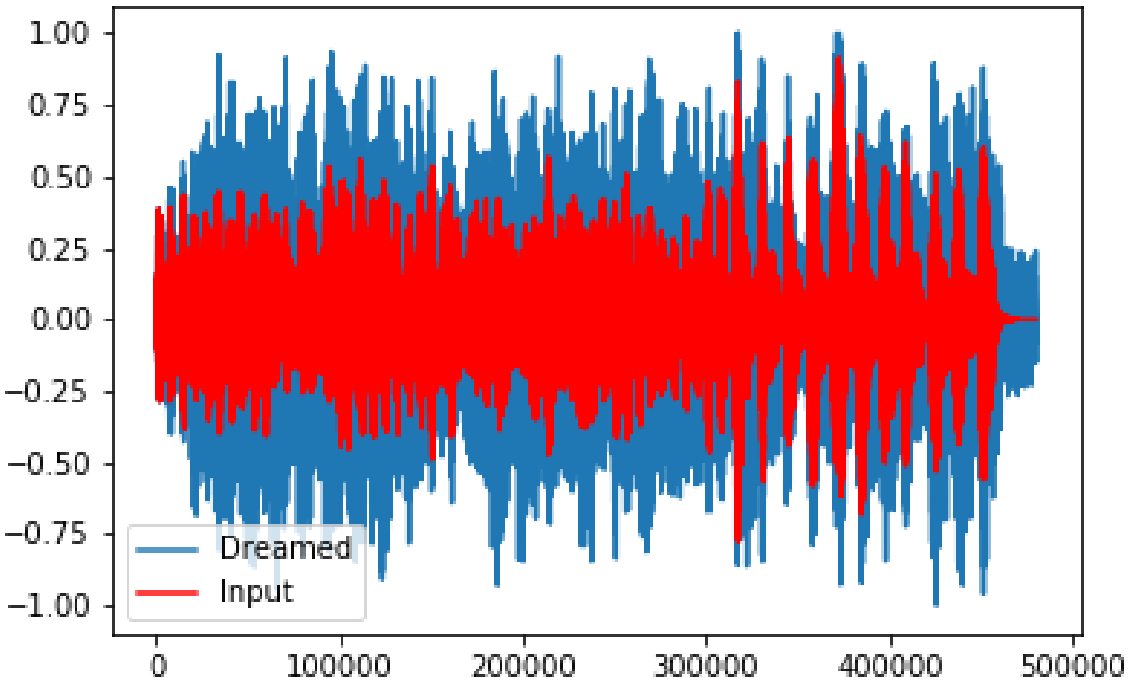

Figure A.7: Visualization of layer 2, filter 6

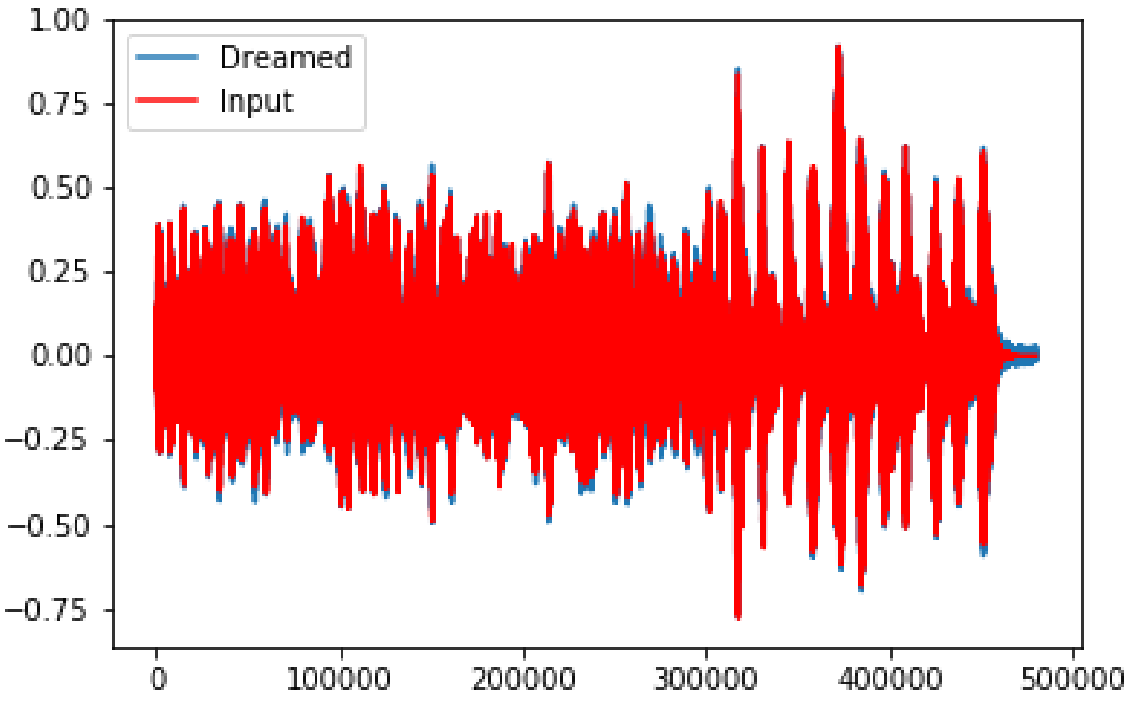

Figure A.8: Visualization of layer 2, filter 7

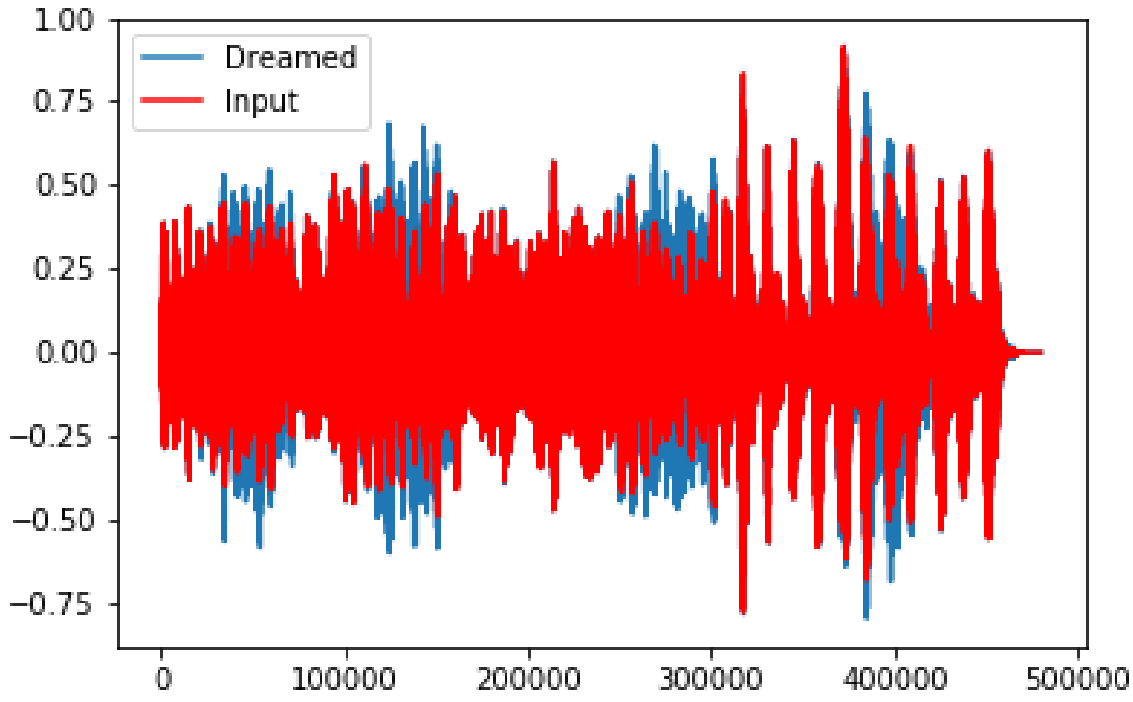

Figure A.9: Visualization of layer 3, filter 0

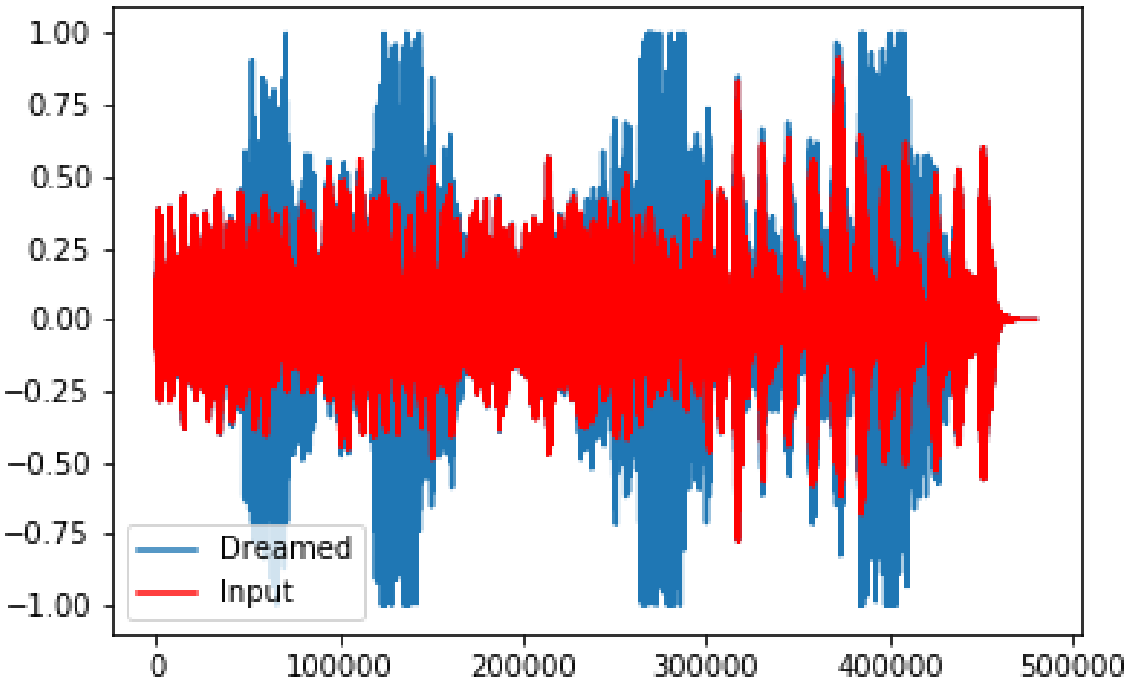

Figure A.10: Visualization of layer 3, filter 1

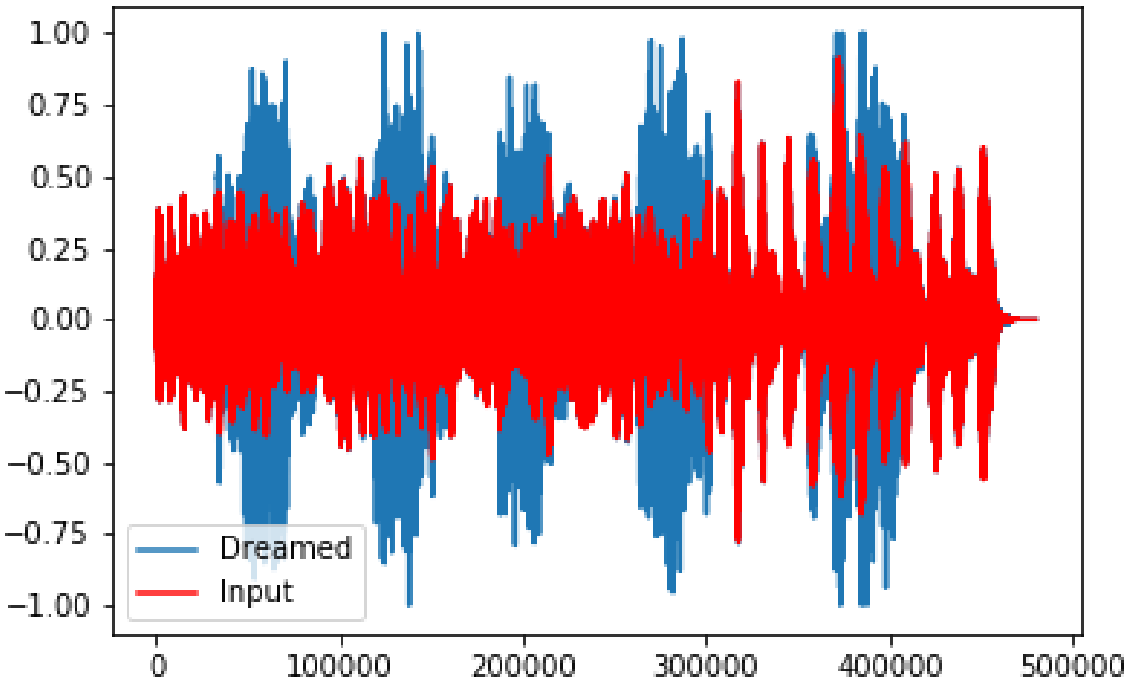

Figure A.11: Visualization of layer 3, filter 2

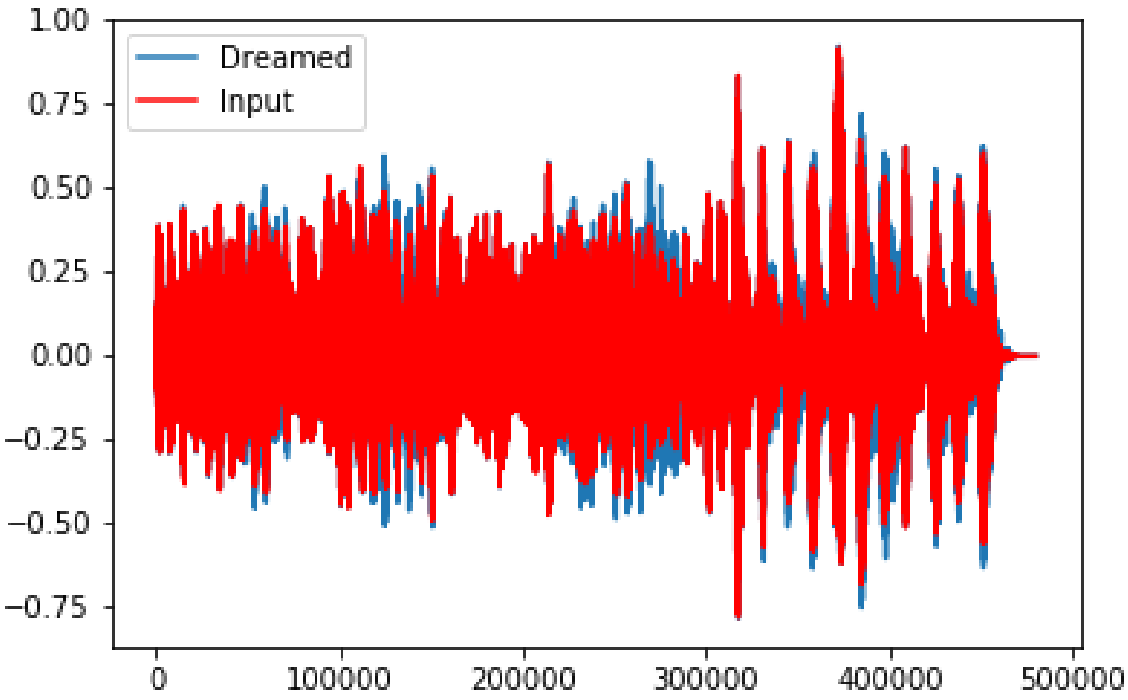

Figure A.12: Visualization of layer 3, filter 3

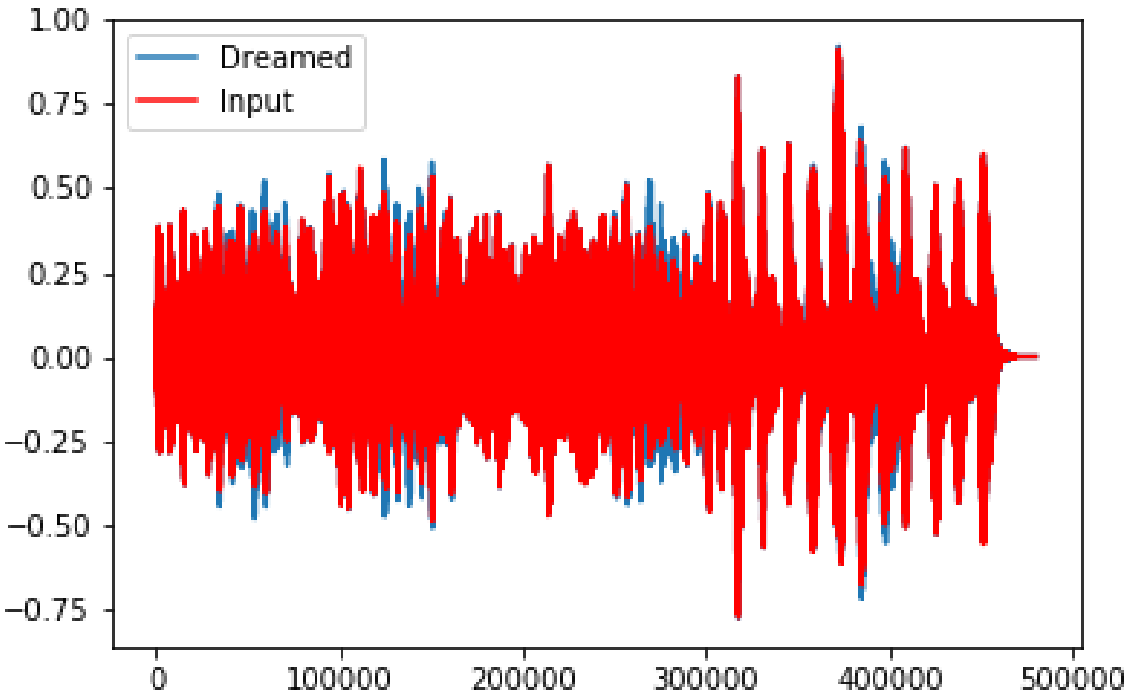

Figure A.13: Visualization of layer 3, filter 4

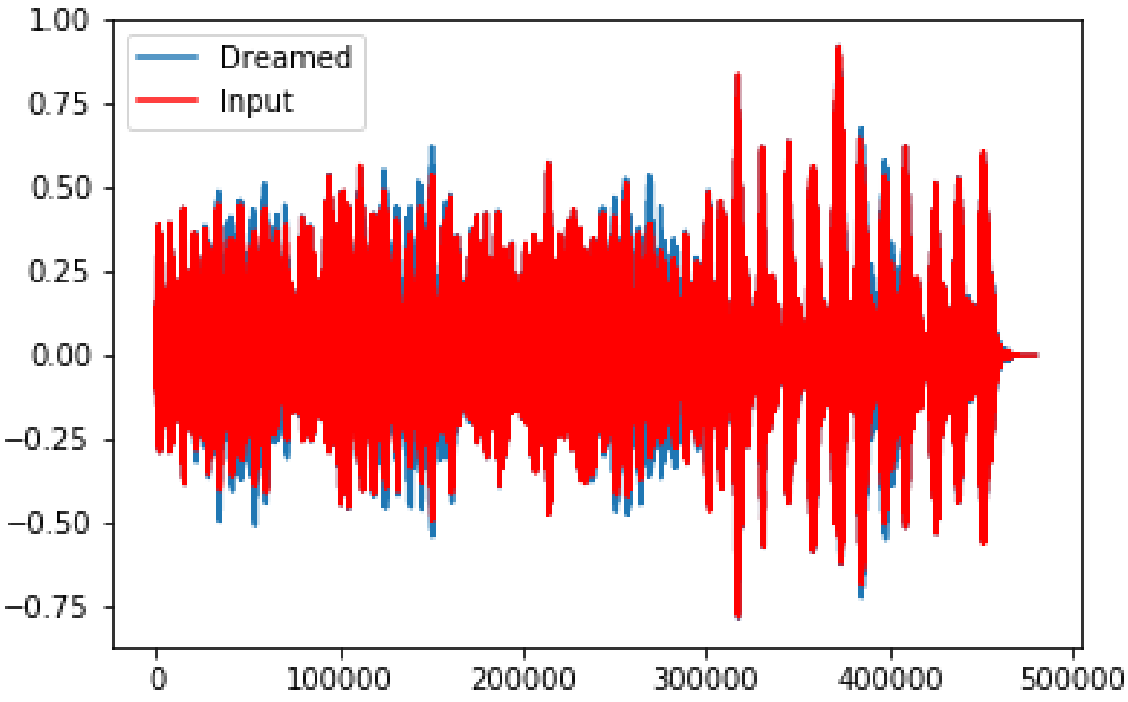

Figure A.14: Visualization of layer 3, filter 5

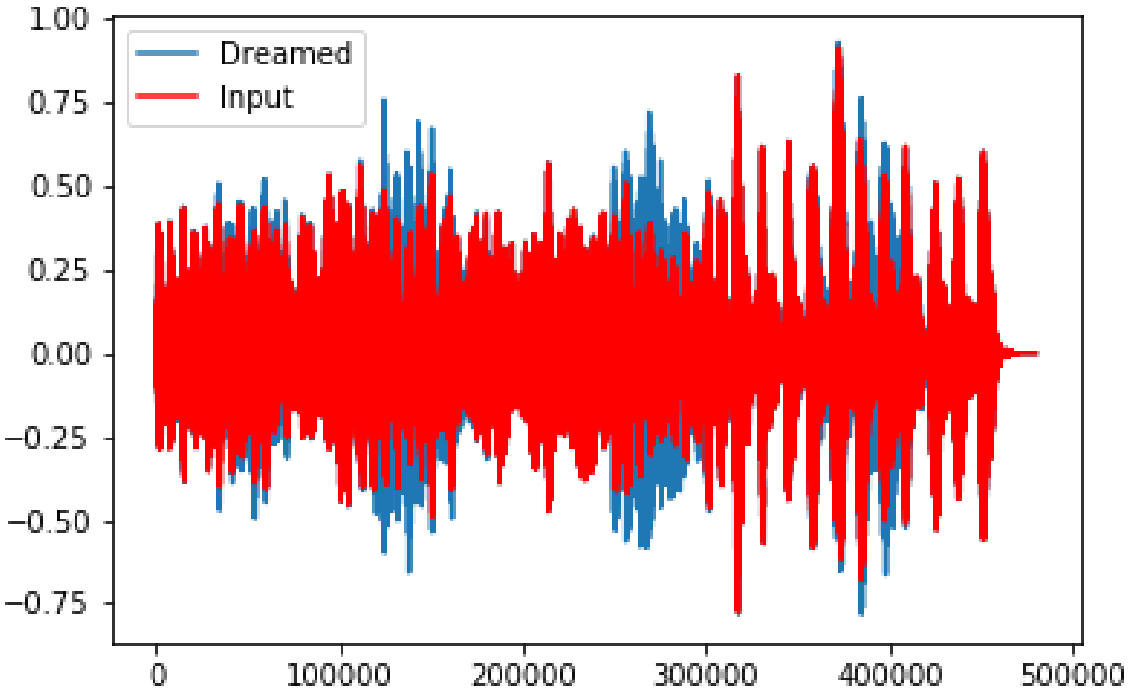

Figure A.15: Visualization of layer 3, filter 6

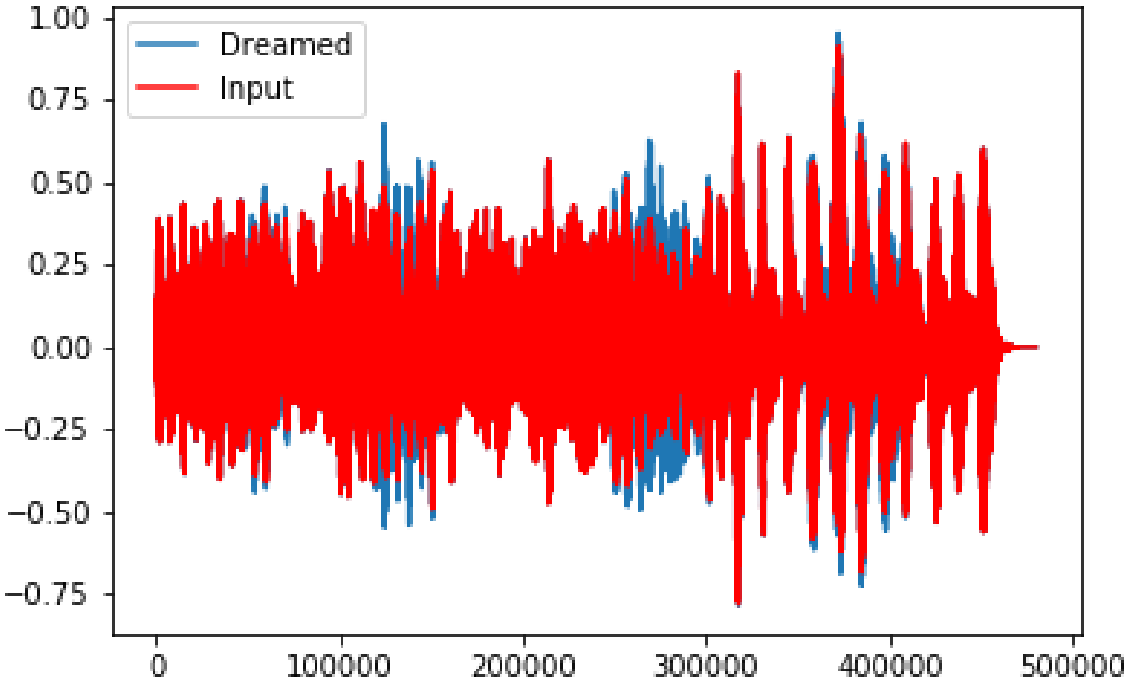

Figure A.16: Visualization of layer 3, filter 7

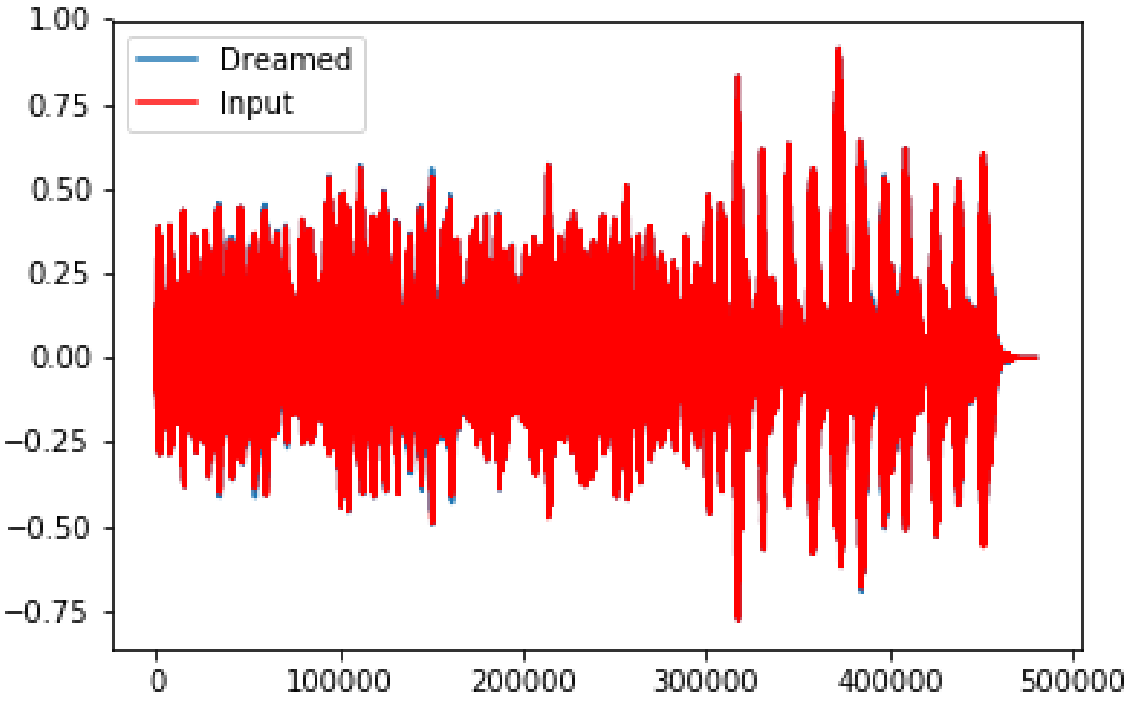

Figure A.17: Visualization of layer 3, filter 8

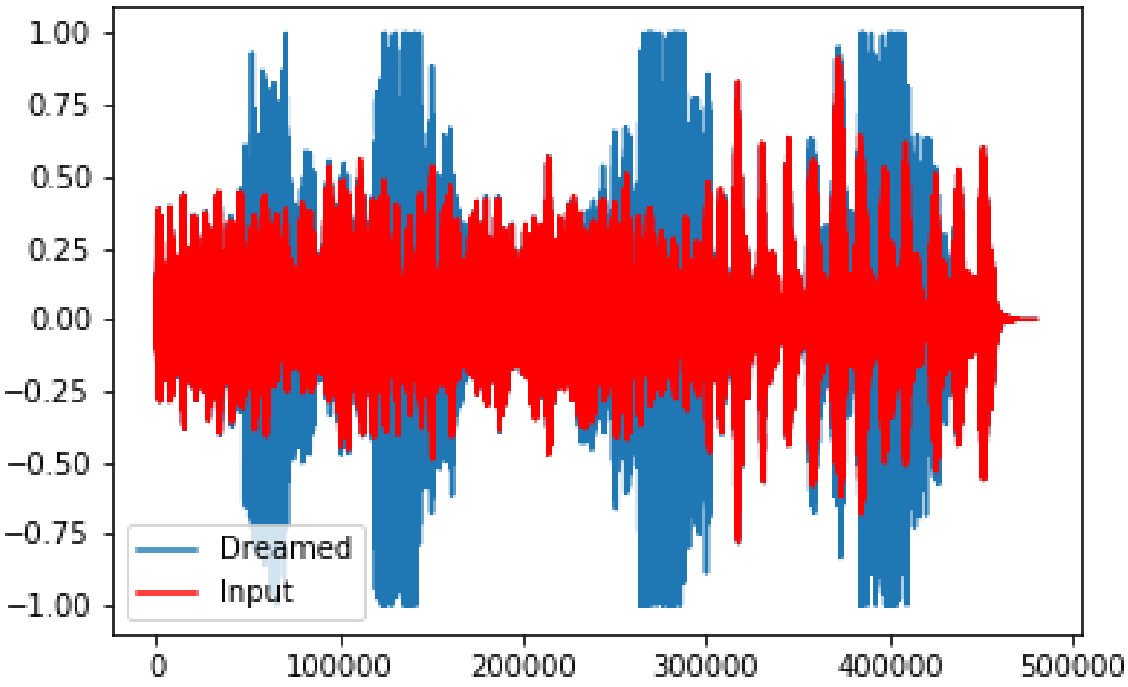

Figure A.18: Visualization of layer 3, filter 9

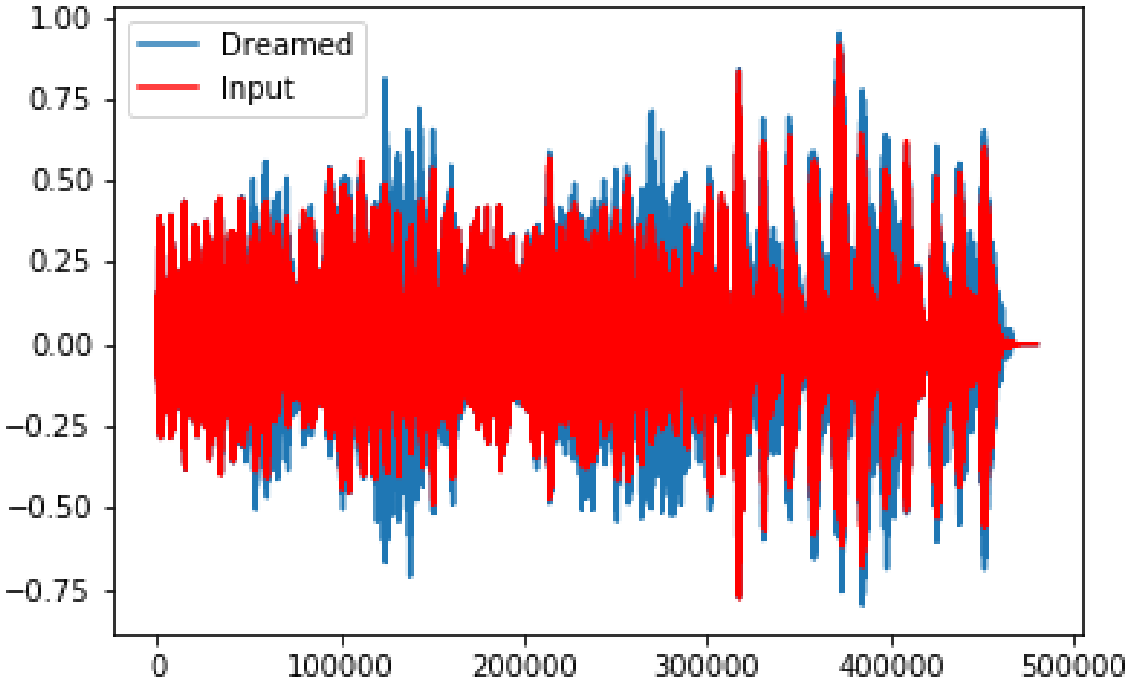

Figure A.19: Visualization of layer 3, filter 10

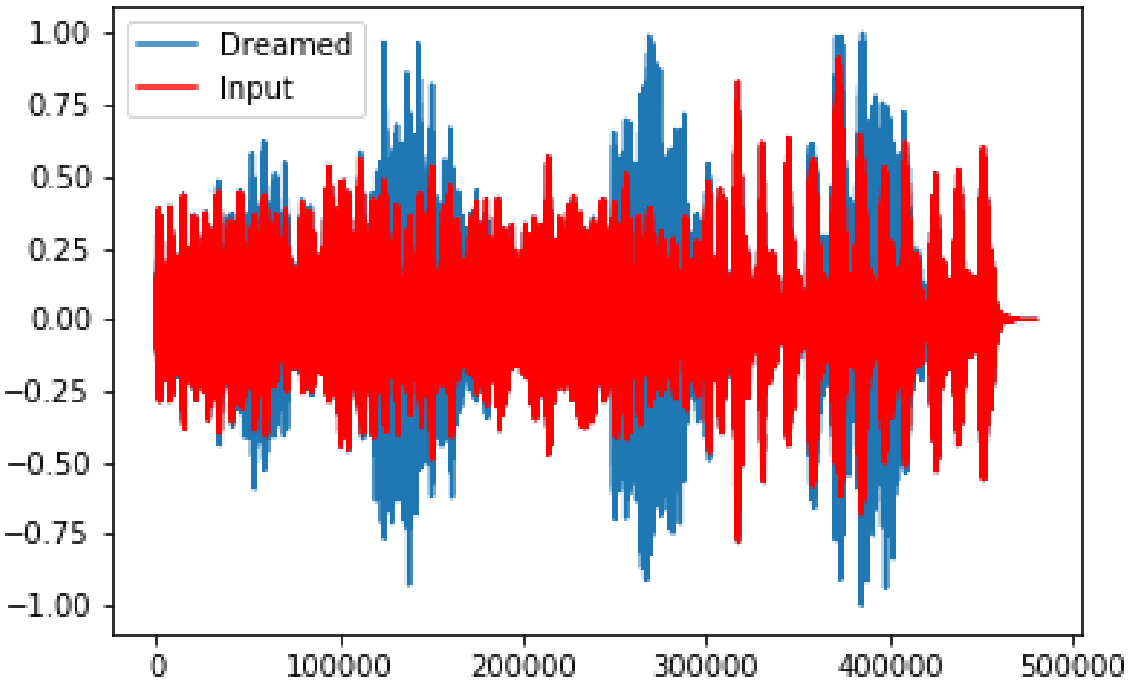

Figure A.20: Visualization of layer 3, filter 11

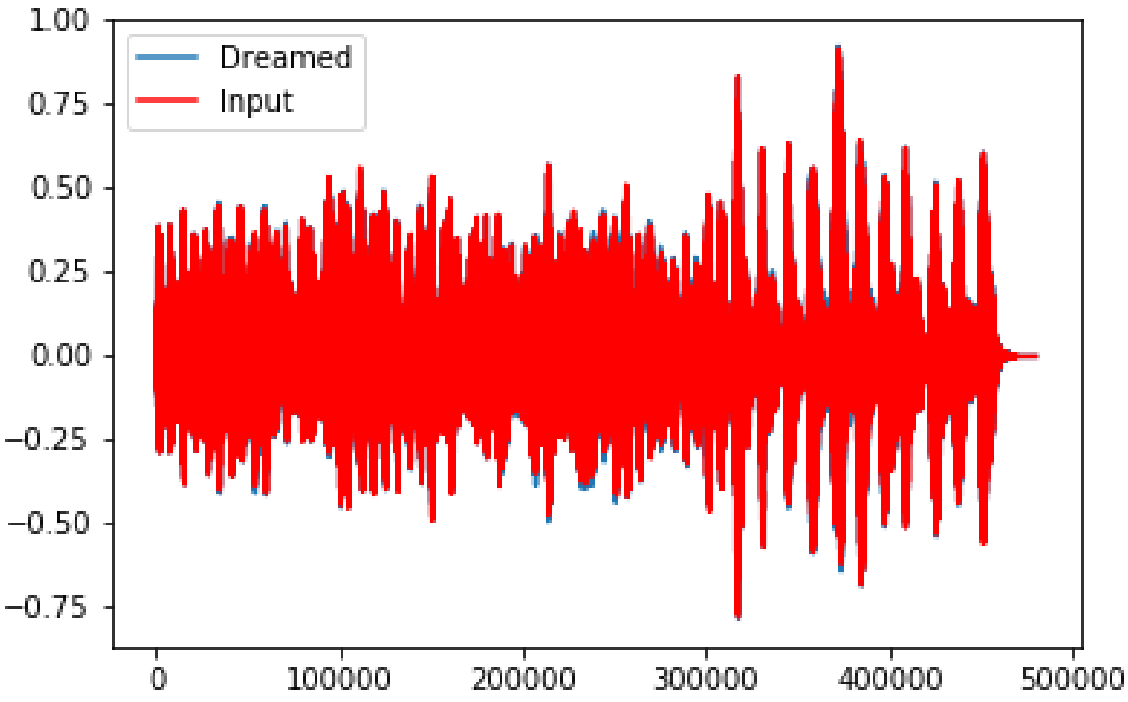

Figure A.21: Visualization of layer 3, filter 12

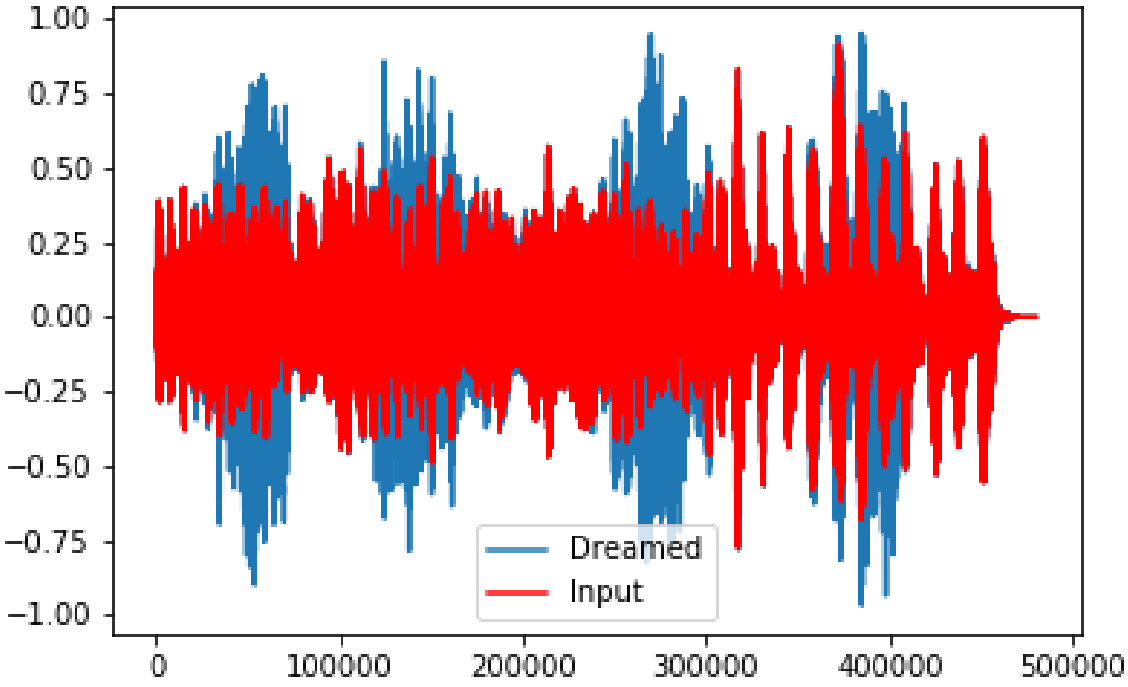

Figure A.22: Visualization of layer 3, filter 13

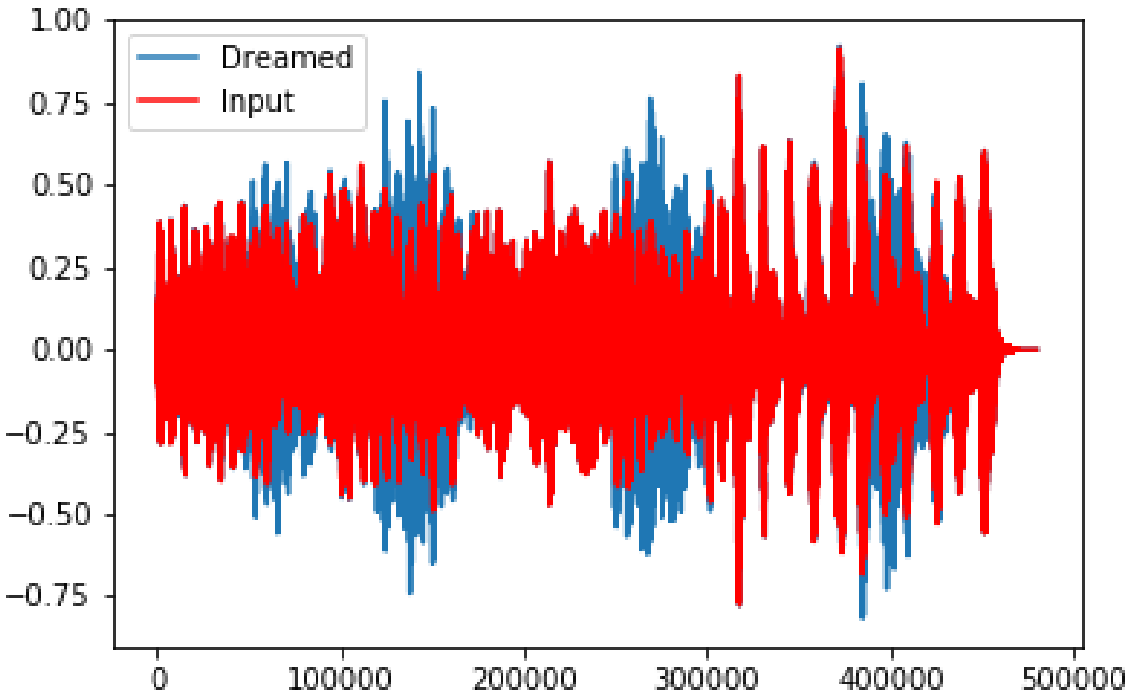

Figure A.23: Visualization of layer 3, filter 14

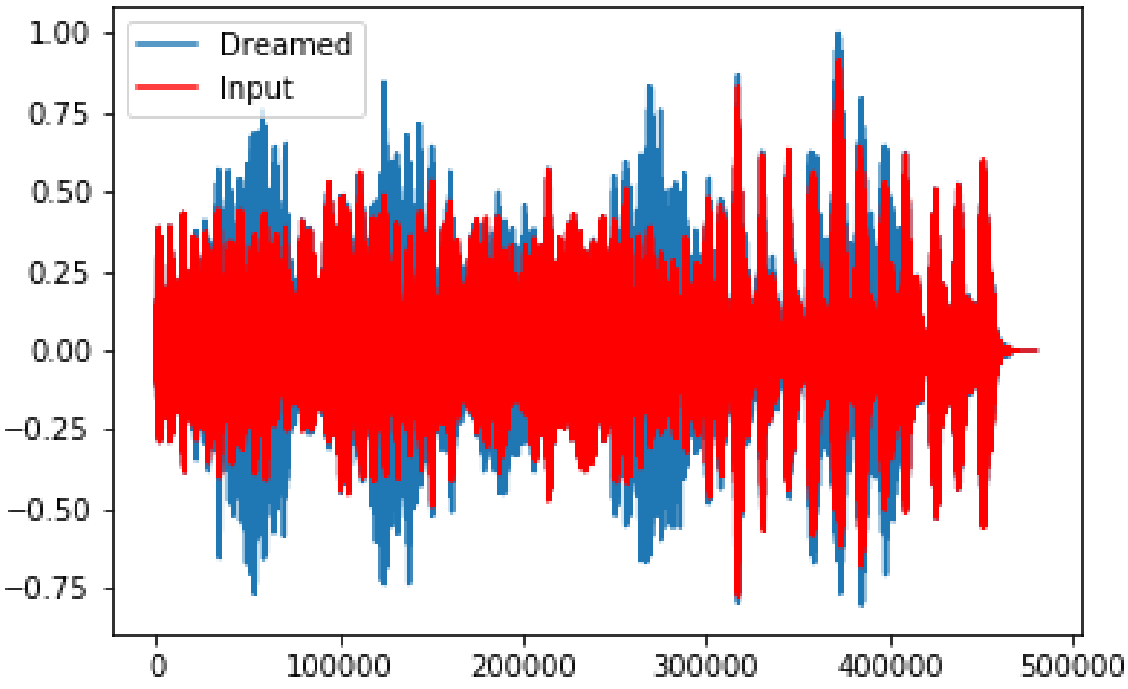

Figure A.24: Visualization of layer 3, filter 15

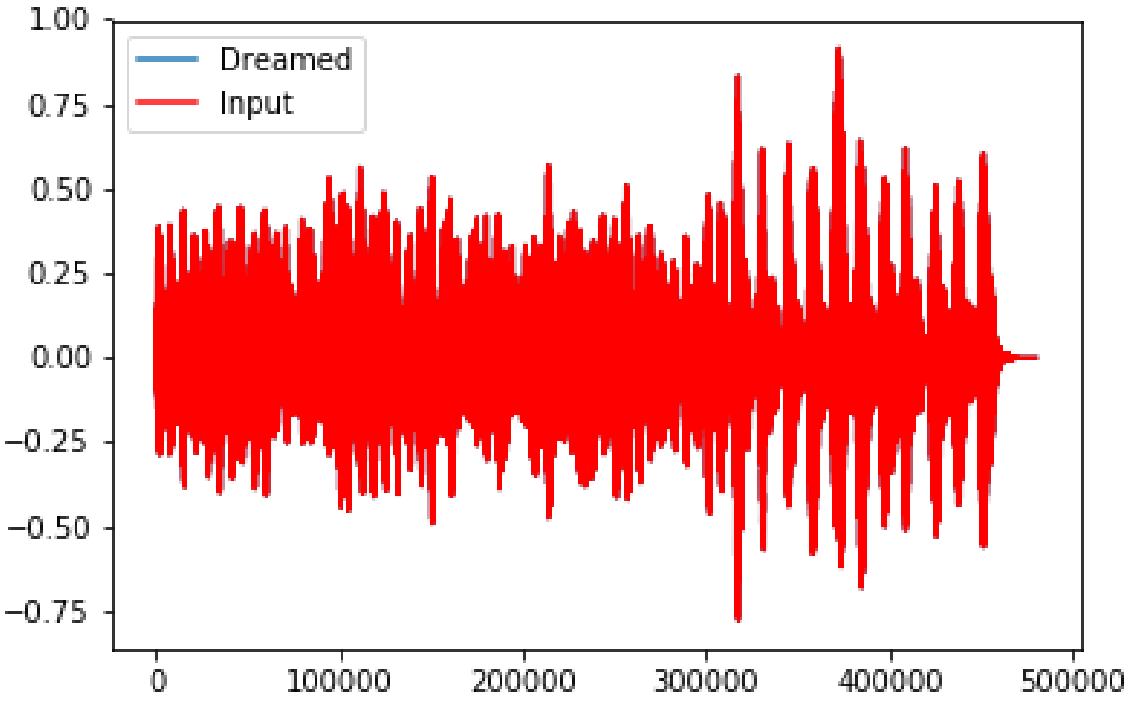

Figure A.25: Visualization of layer 4, filter 0

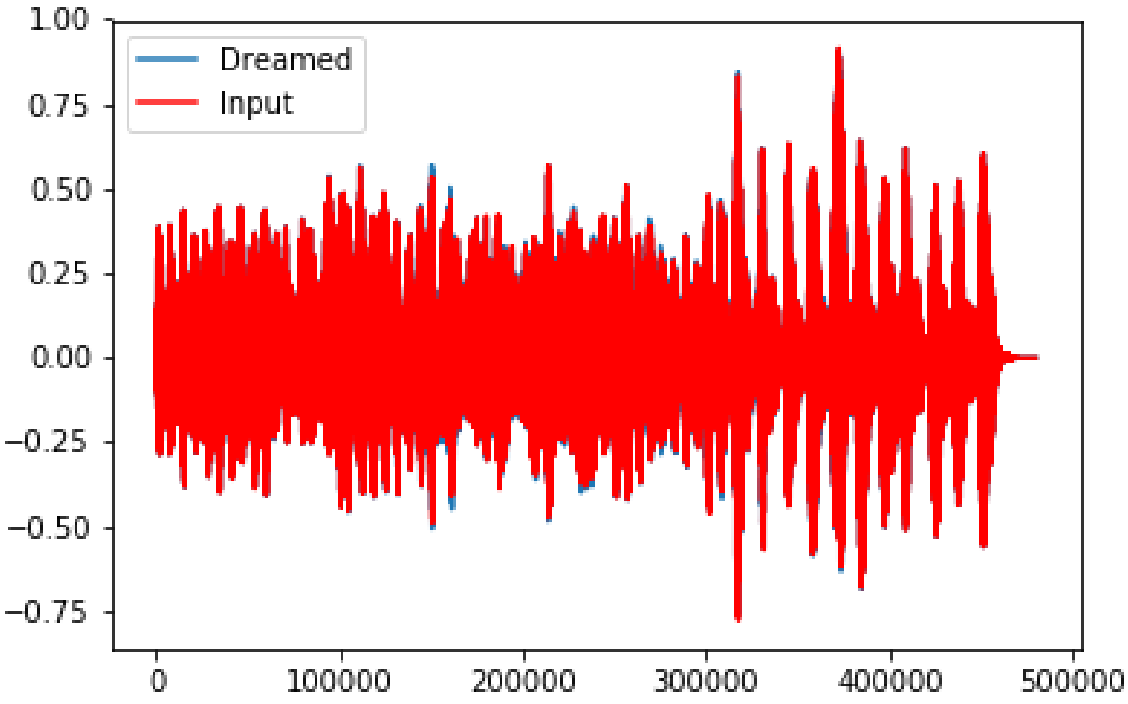

Figure A.26: Visualization of layer 4, filter 1

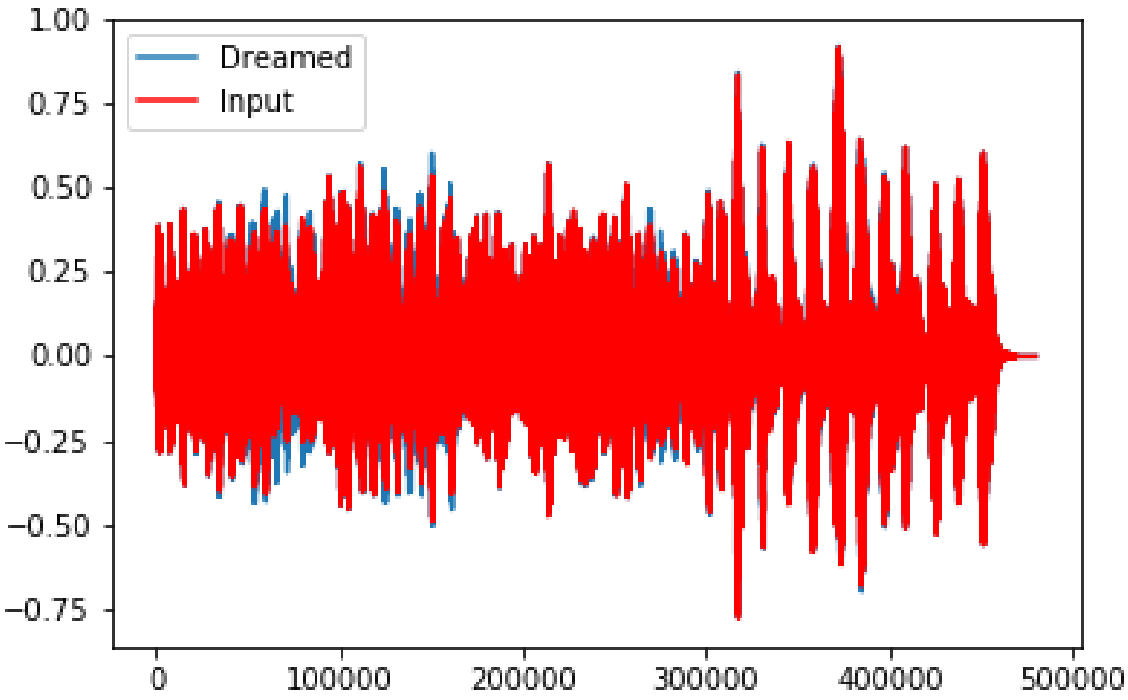

Figure A.27: Visualization of layer 4, filter 2

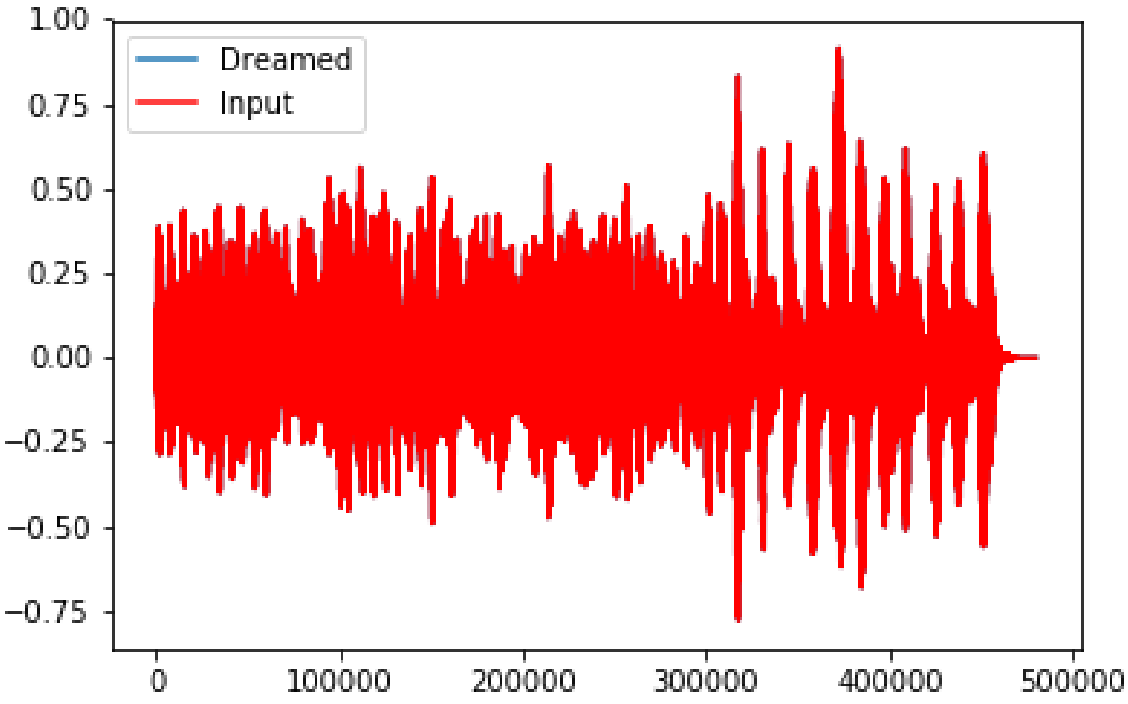

Figure A.28: Visualization of layer 4, filter 3

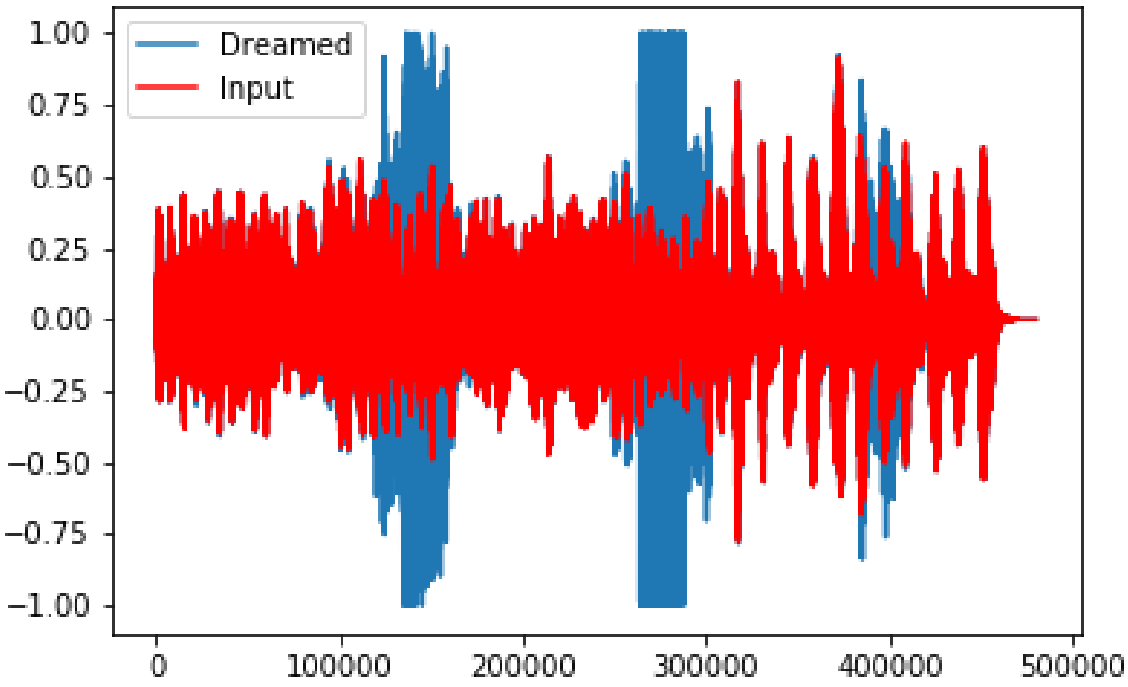

Figure A.29: Visualization of layer 4, filter 4

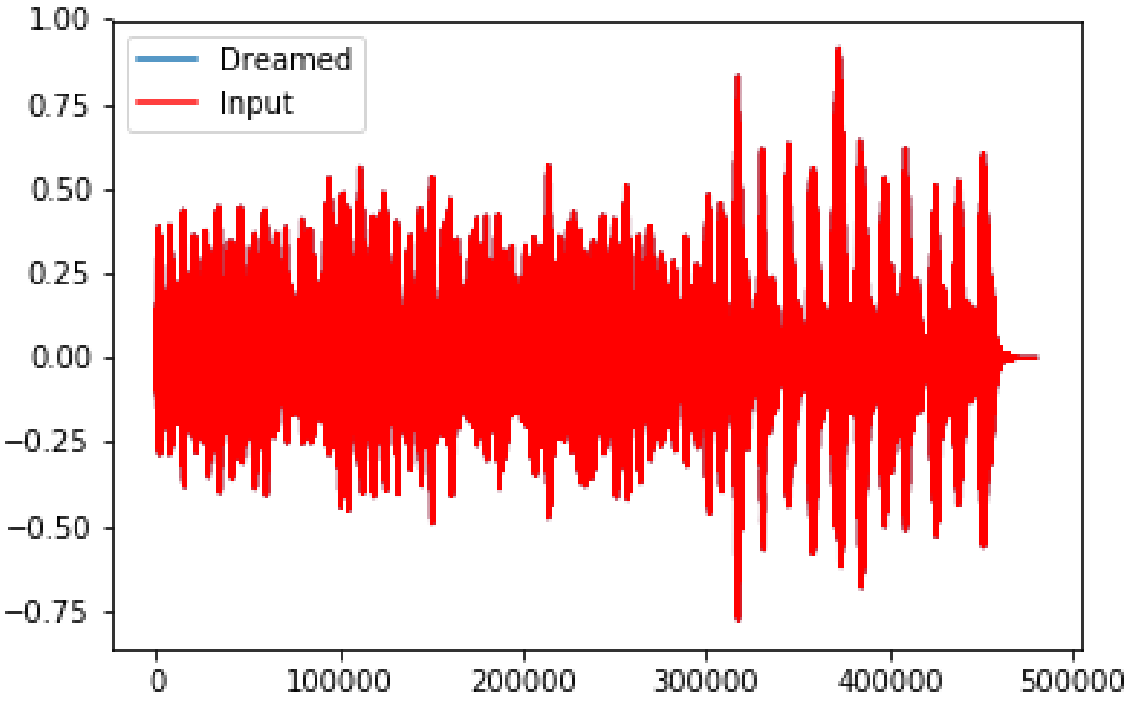

Figure A.30: Visualization of layer 4, filter 5

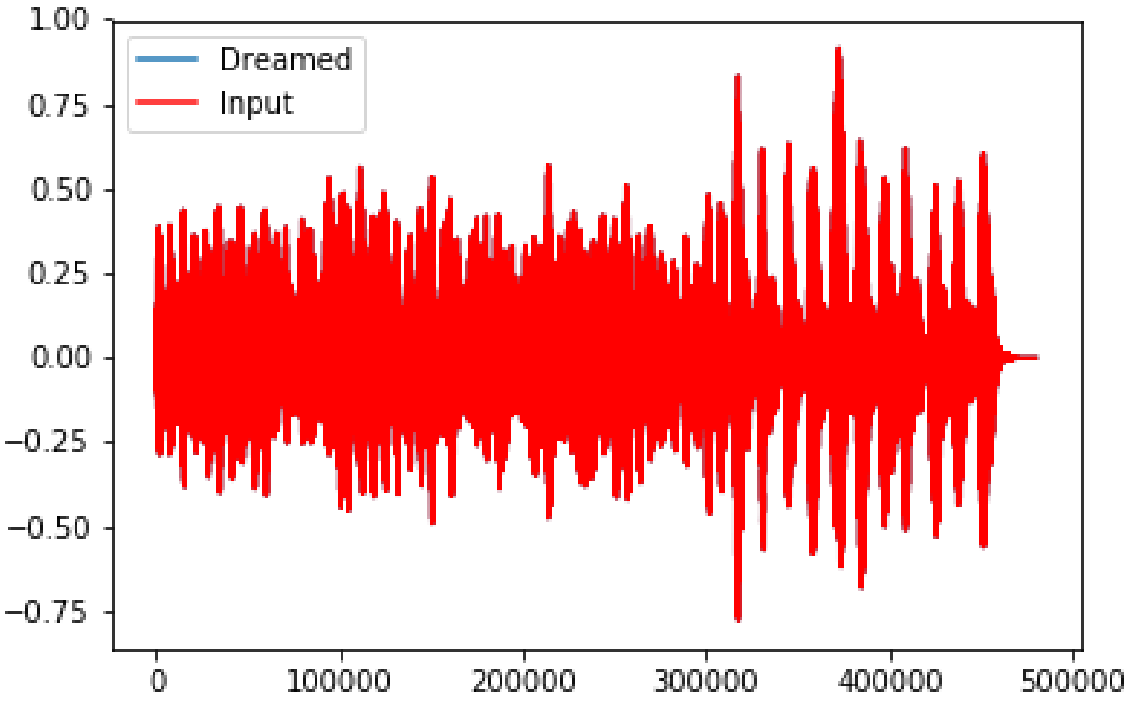

Figure A.31: Visualization of layer 4, filter 6

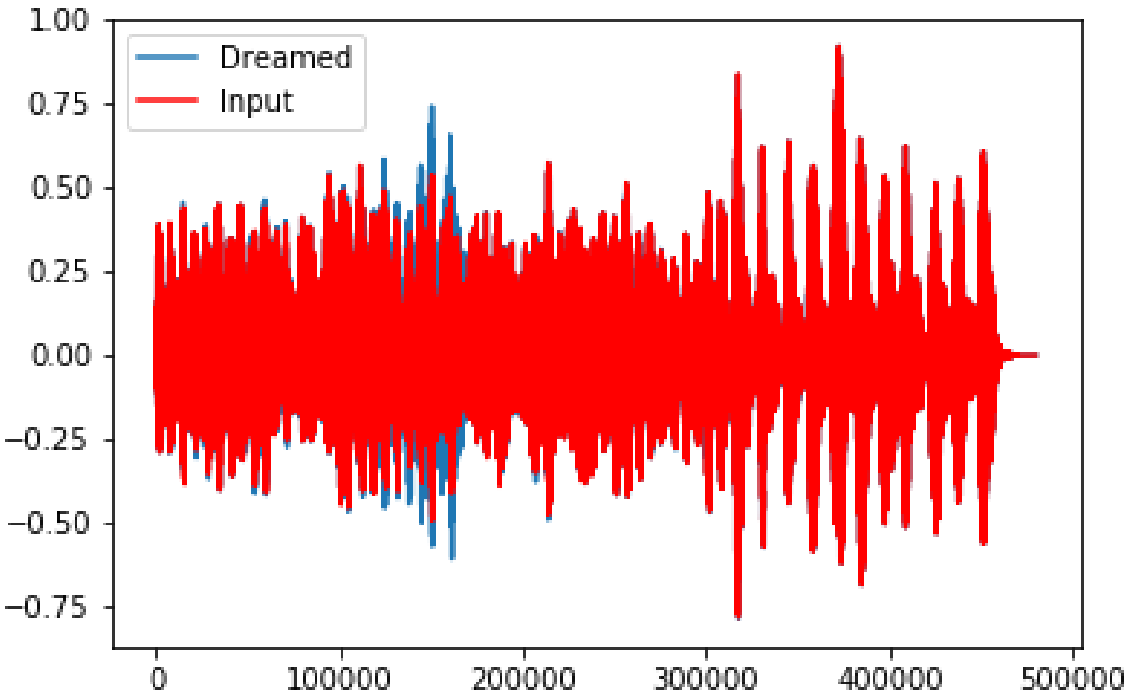

Figure A.32: Visualization of layer 4, filter 7

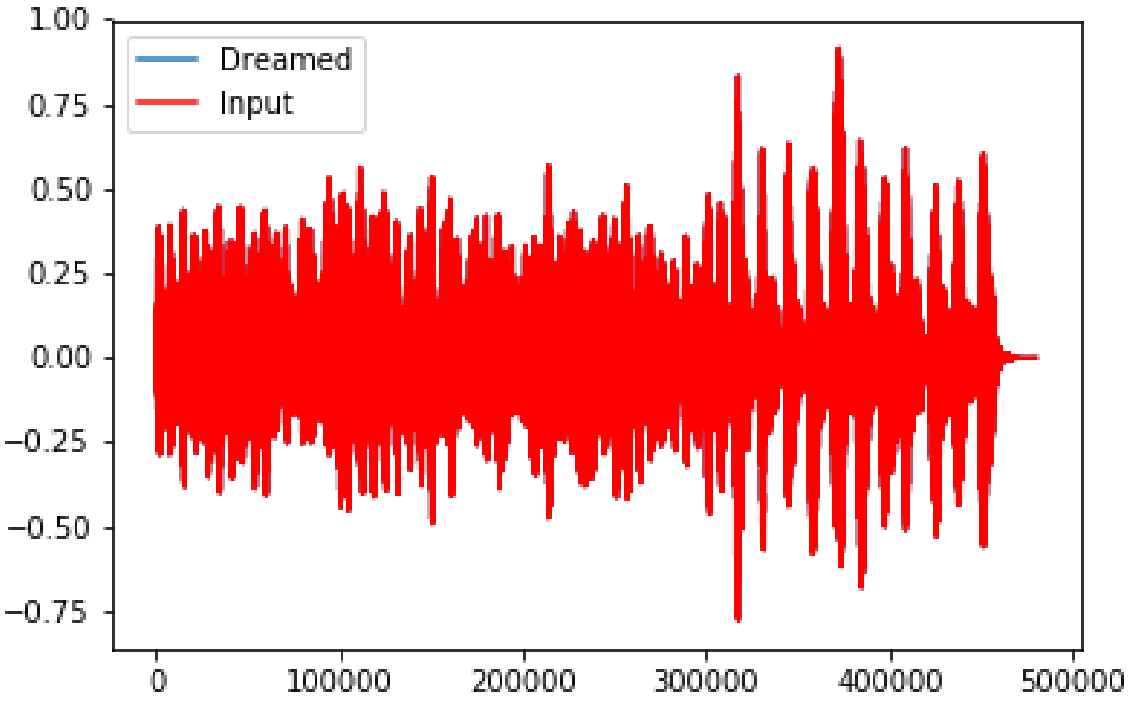

Figure A.33: Visualization of layer 4, filter 8

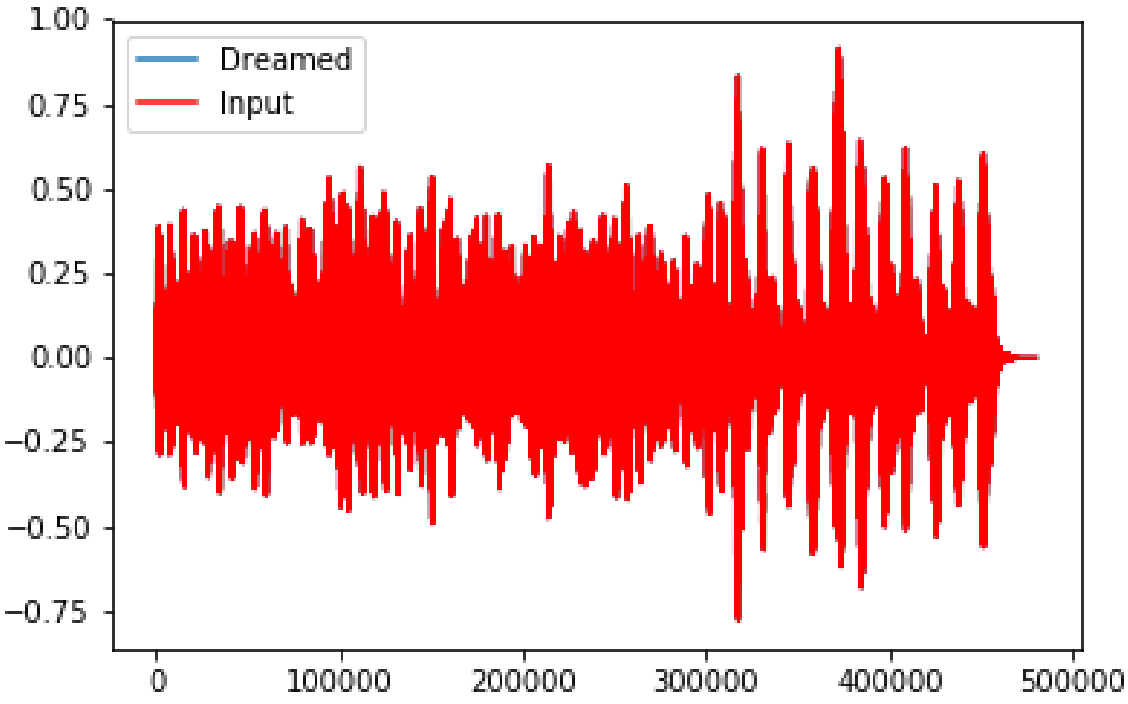

Figure A.34: Visualization of layer 4, filter 9

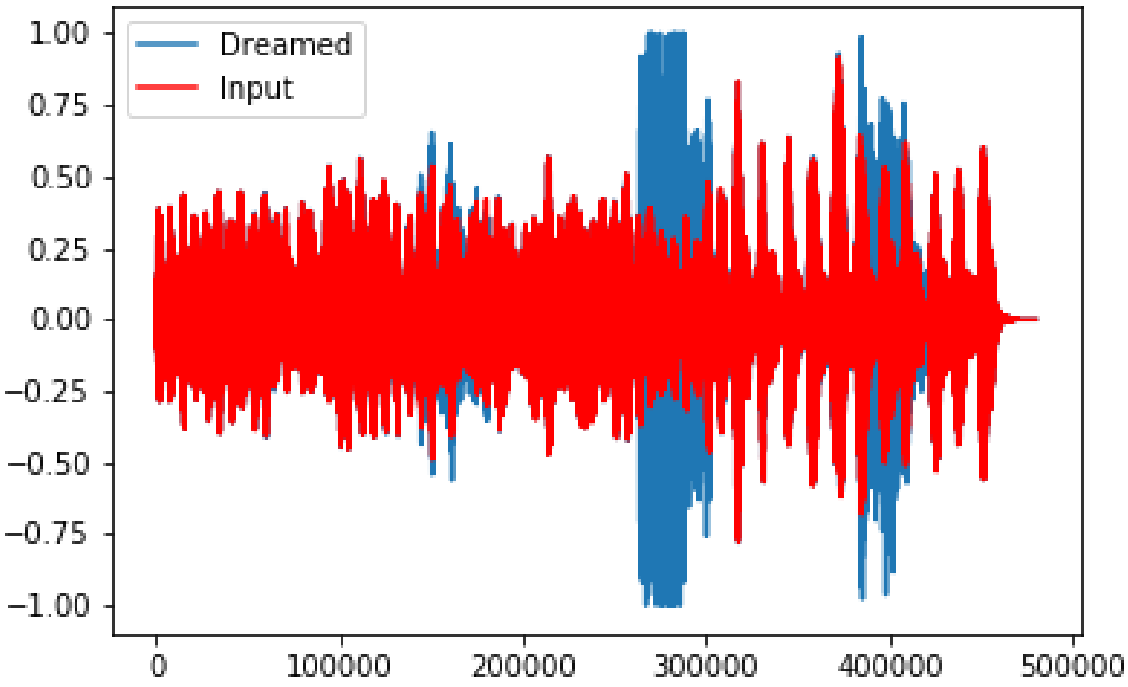

Figure A.35: Visualization of layer 4, filter 10

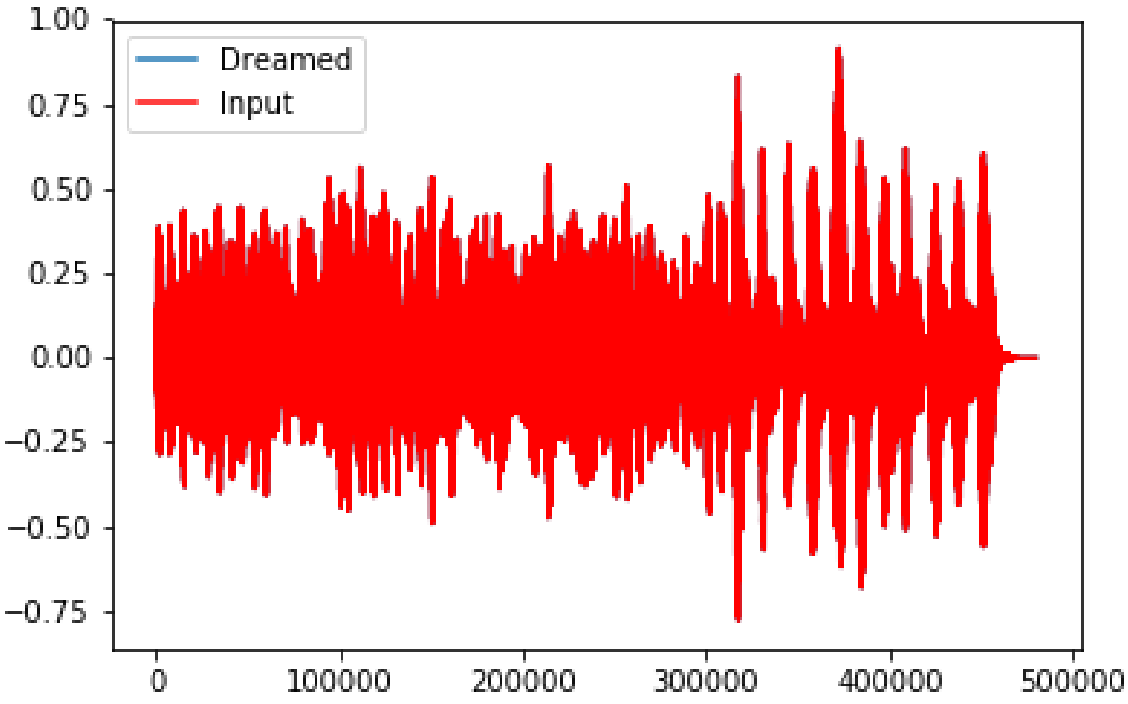

Figure A.36: Visualization of layer 4, filter 11

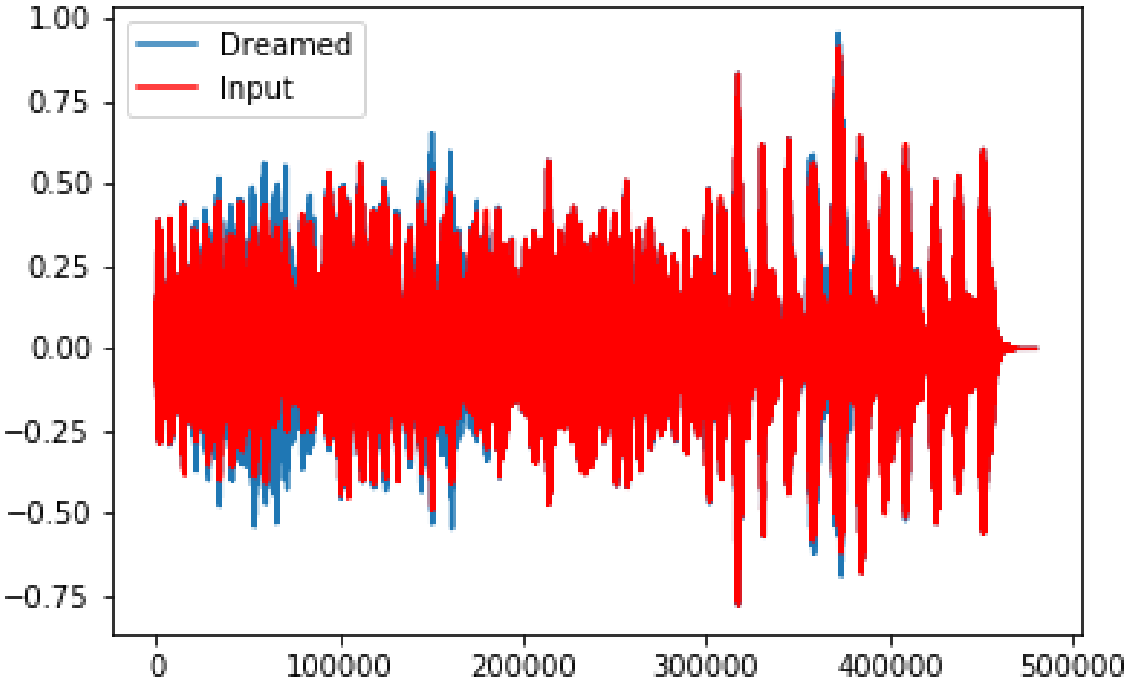

Figure A.37: Visualization of layer 4, filter 12

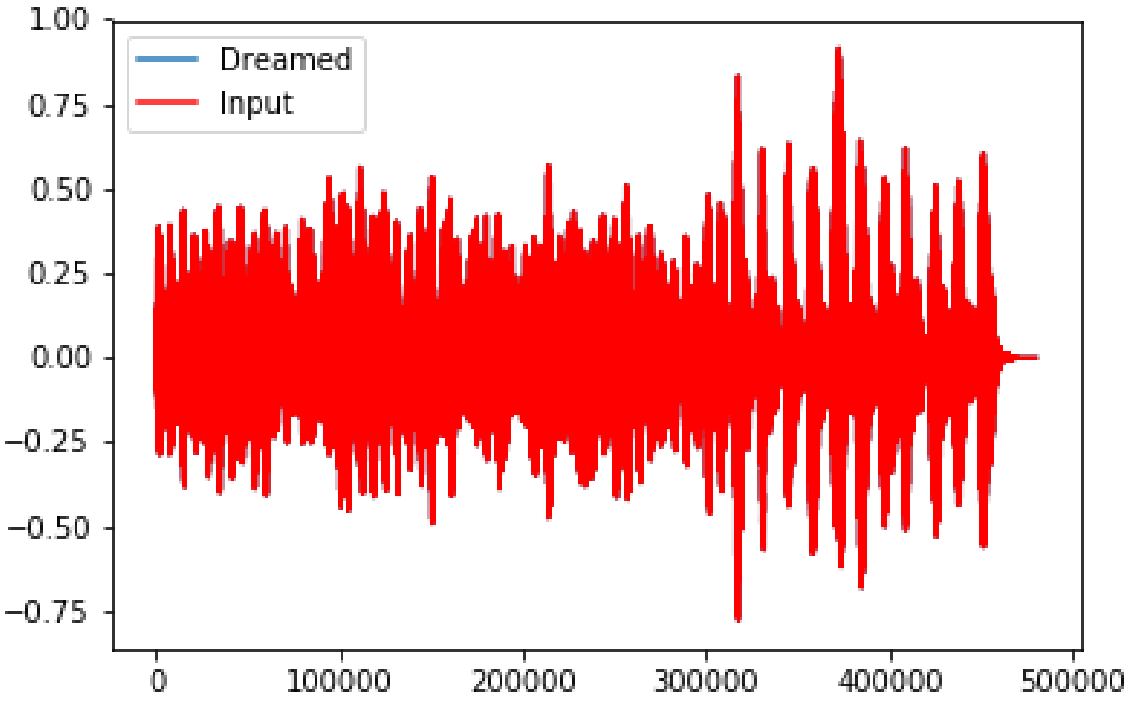

Figure A.38: Visualization of layer 4, filter 13

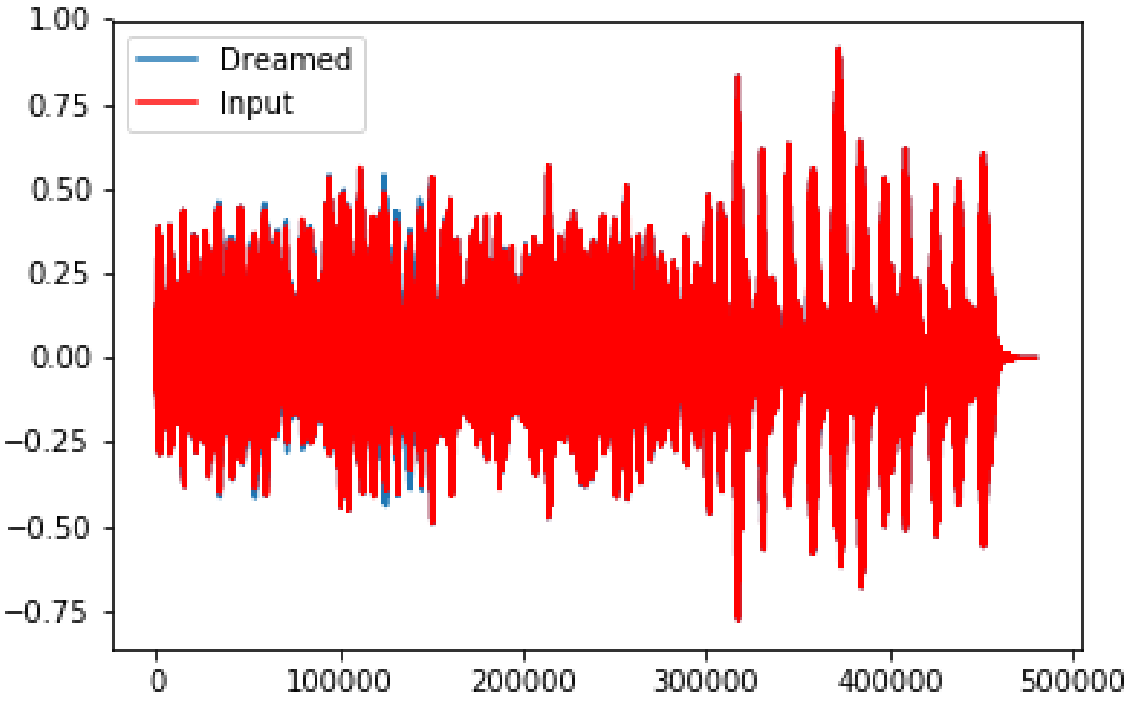

Figure A.39: Visualization of layer 4, filter 14

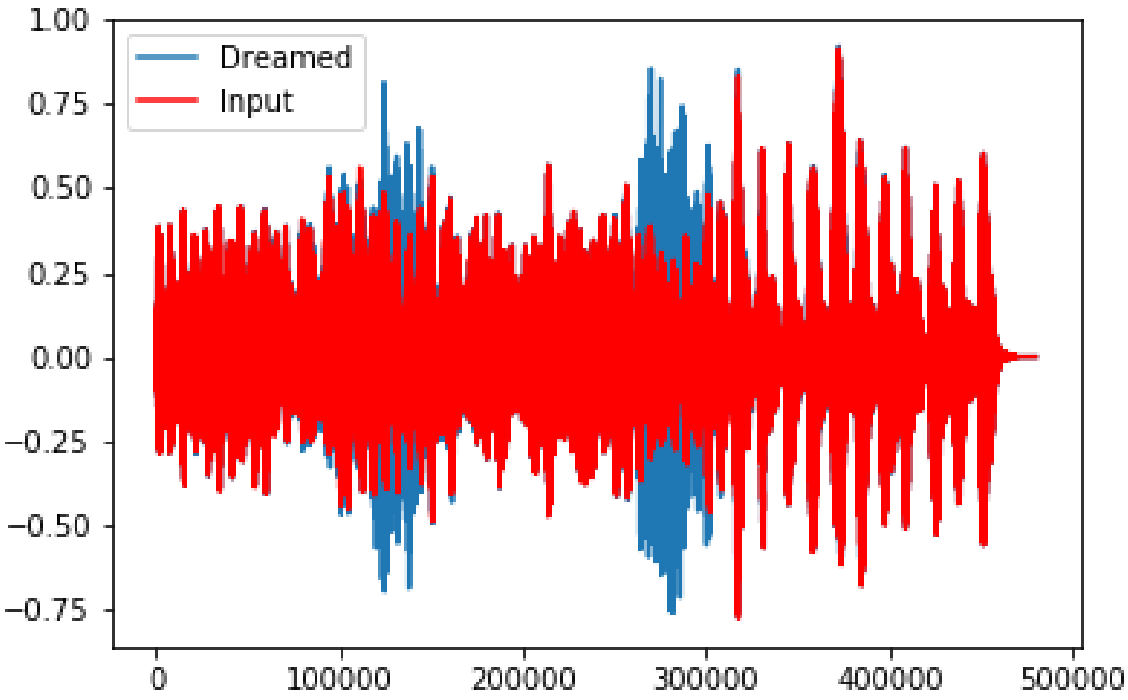

Figure A.40: Visualization of layer 4, filter 15

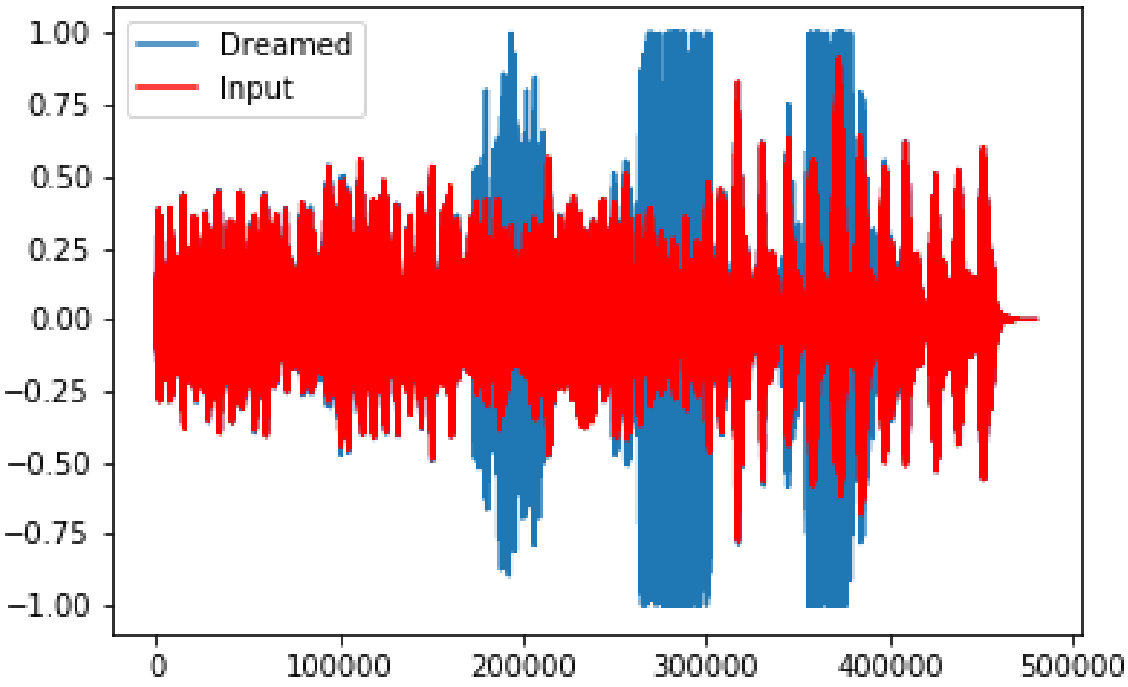

Figure A.41: Visualization of layer 4, filter 16

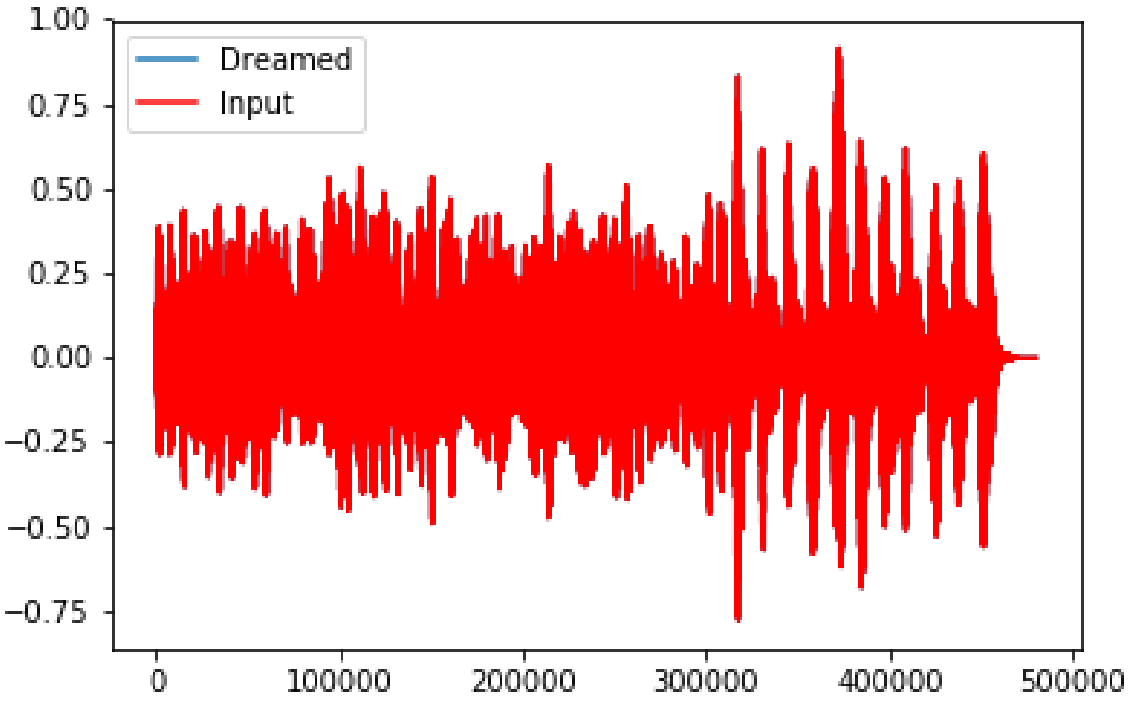

Figure A.42: Visualization of layer 4, filter 17

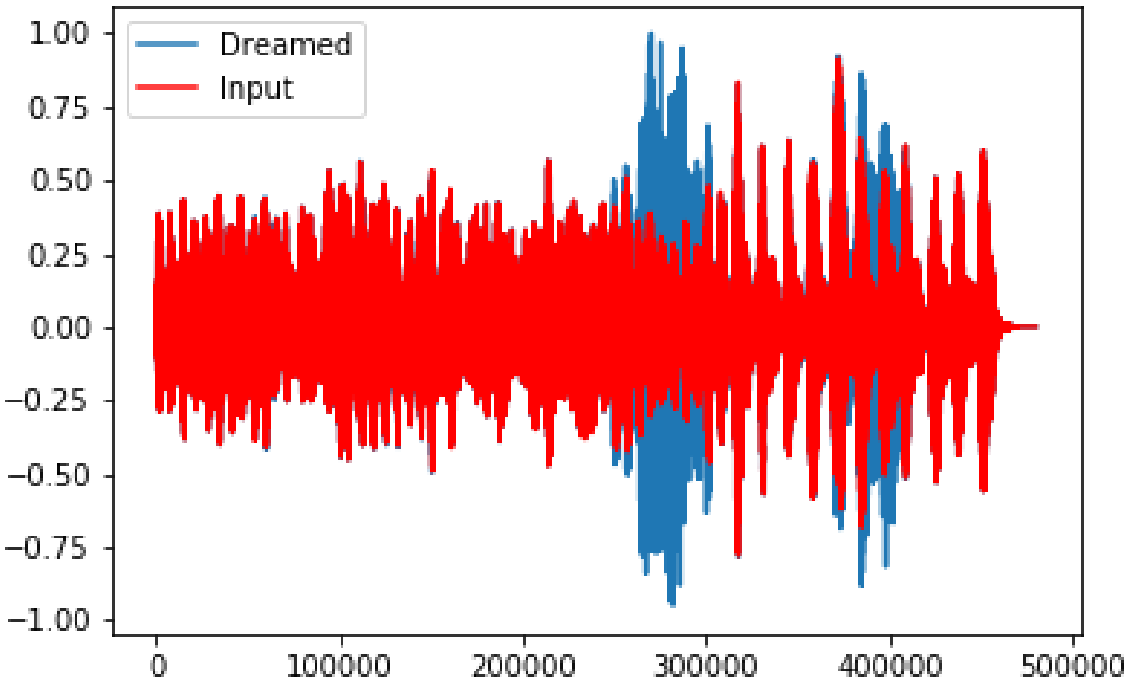

Figure A.43: Visualization of layer 4, filter 18

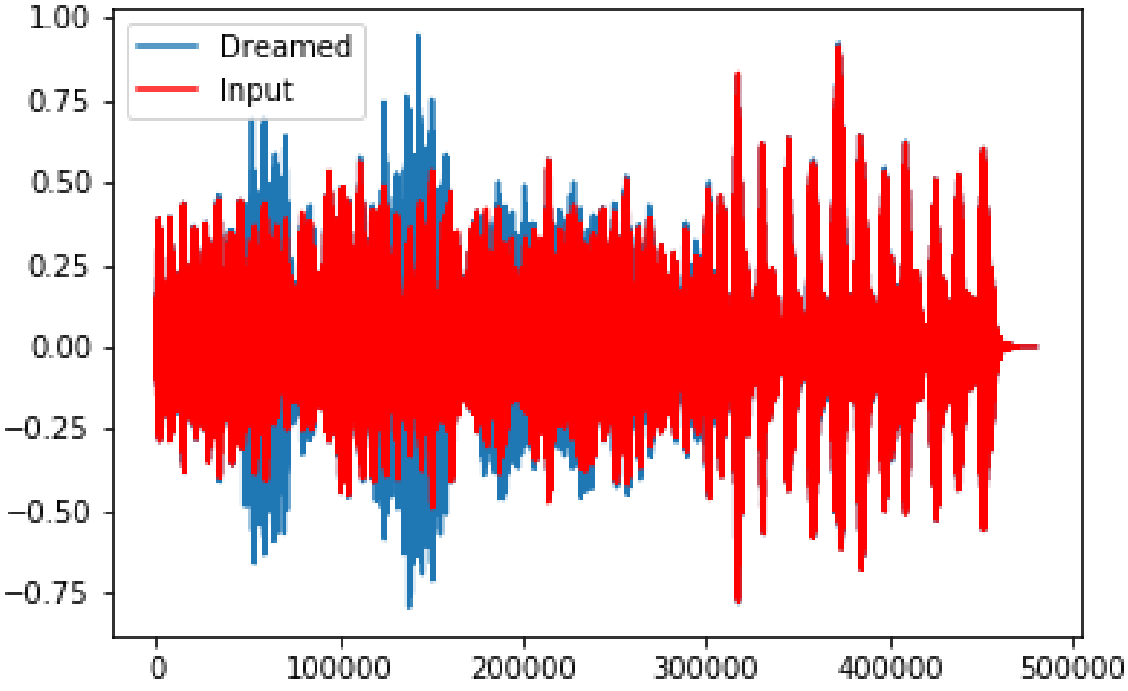

Figure A.44: Visualization of layer 4, filter 19

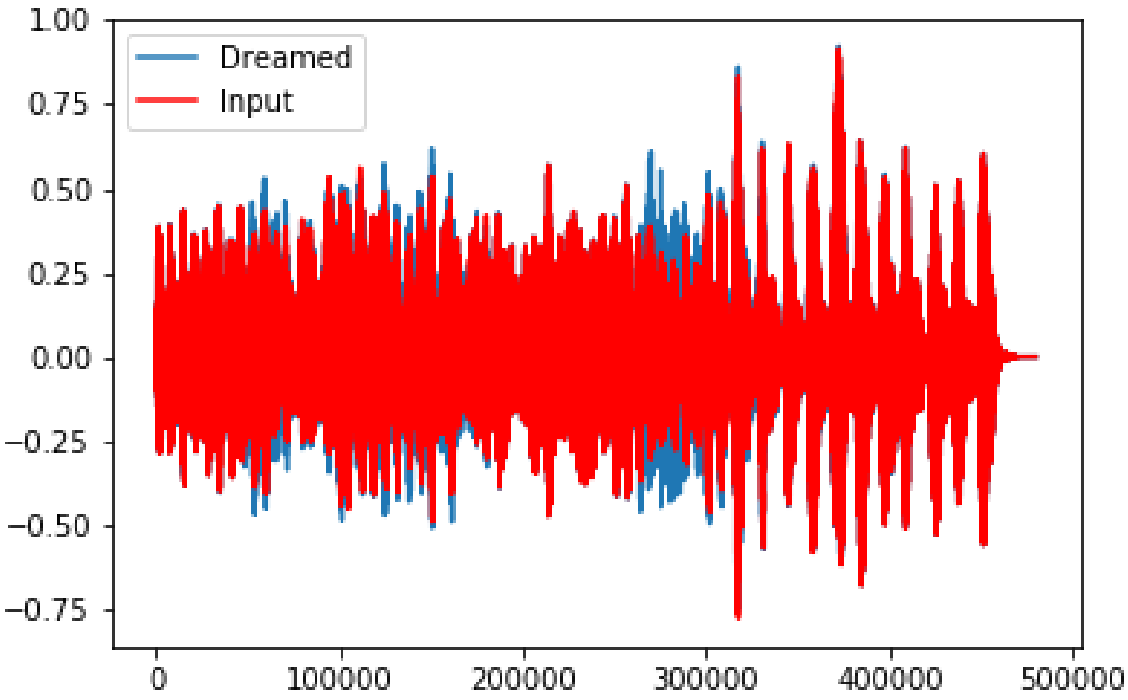

Figure A.45: Visualization of layer 4, filter 20

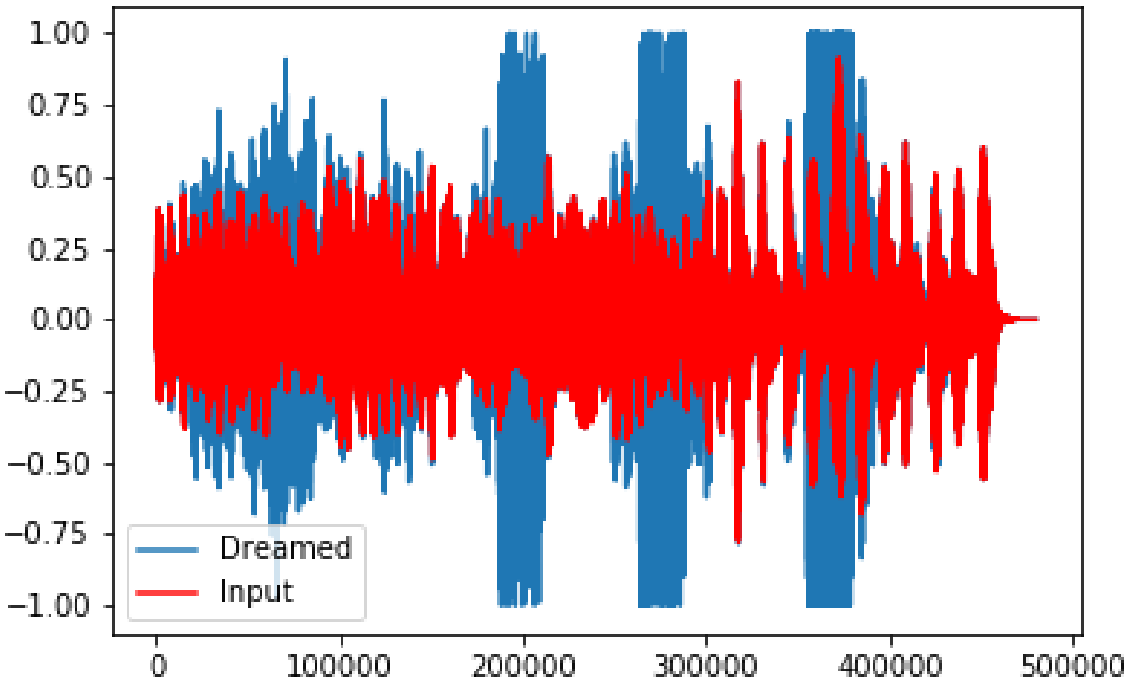

Figure A.46: Visualization of layer 4, filter 21

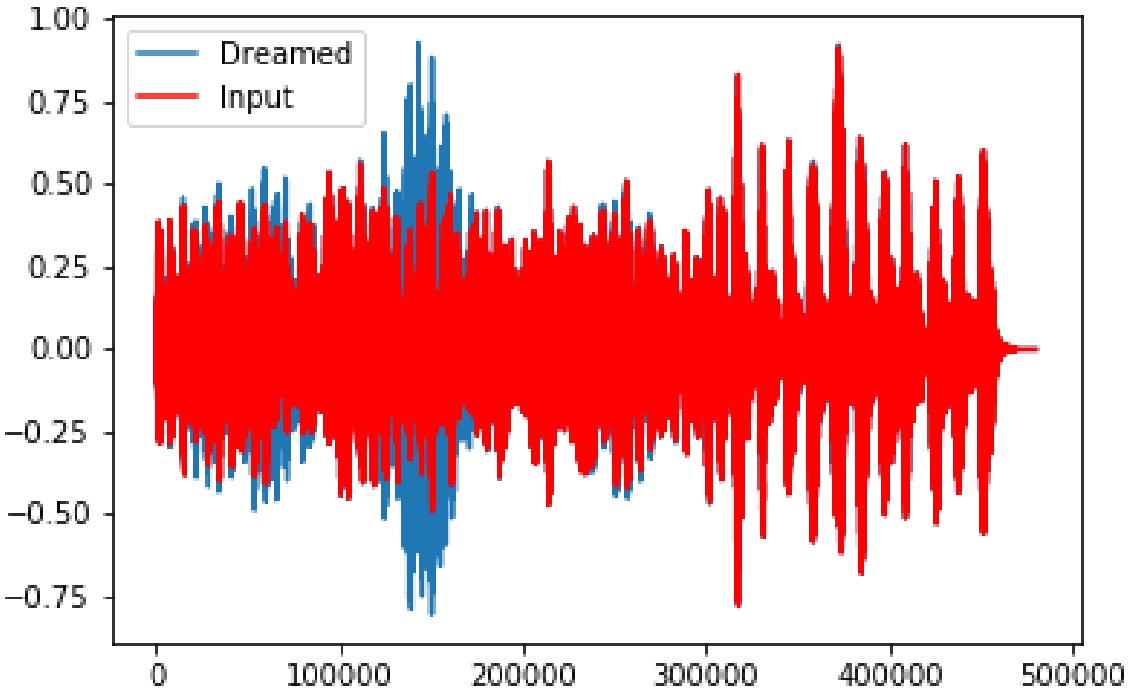

Figure A.47: Visualization of layer 4, filter 22

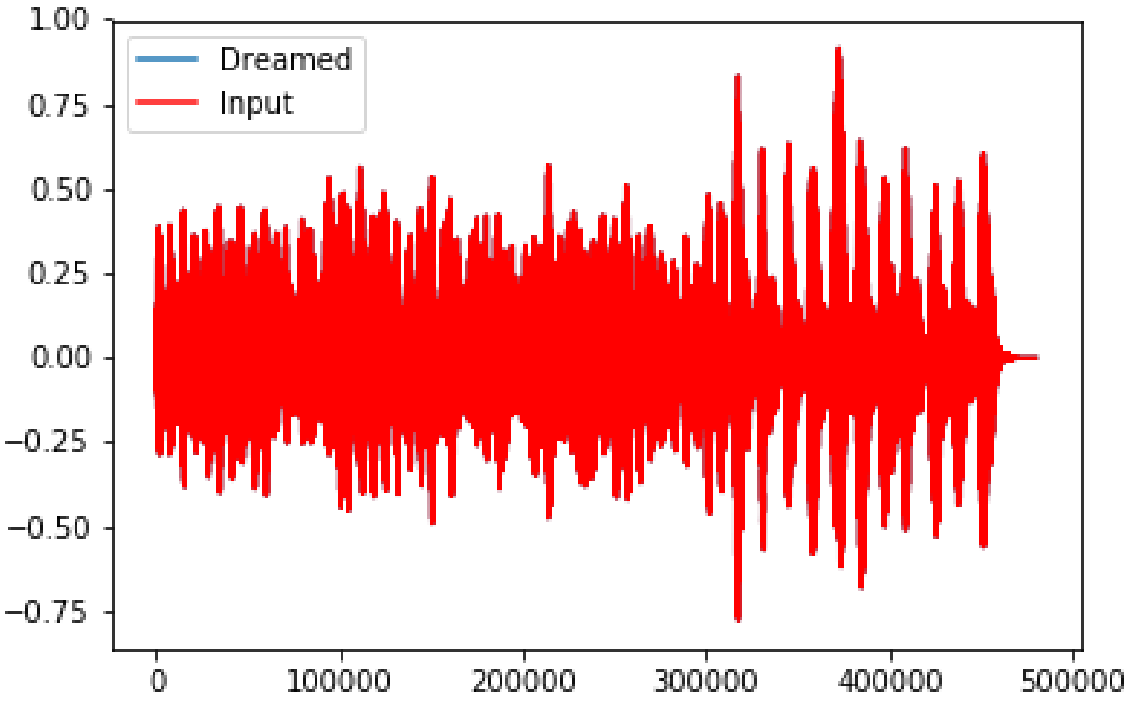

Figure A.48: Visualization of layer 4, filter 23

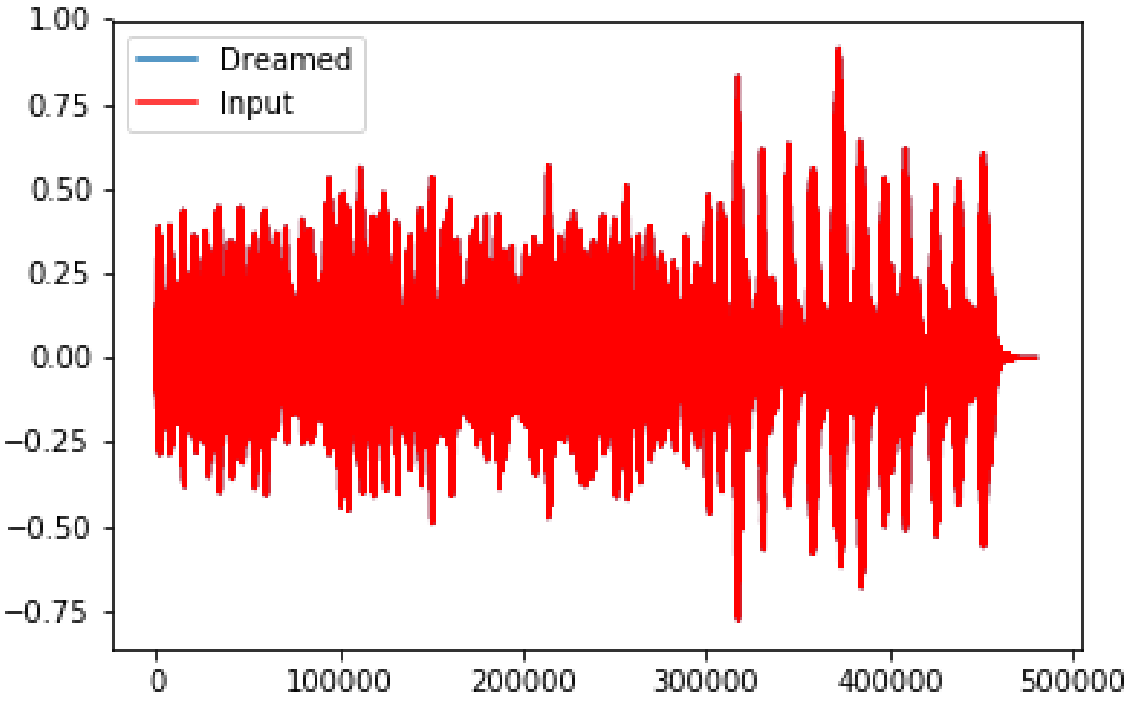

Figure A.49: Visualization of layer 4, filter 24

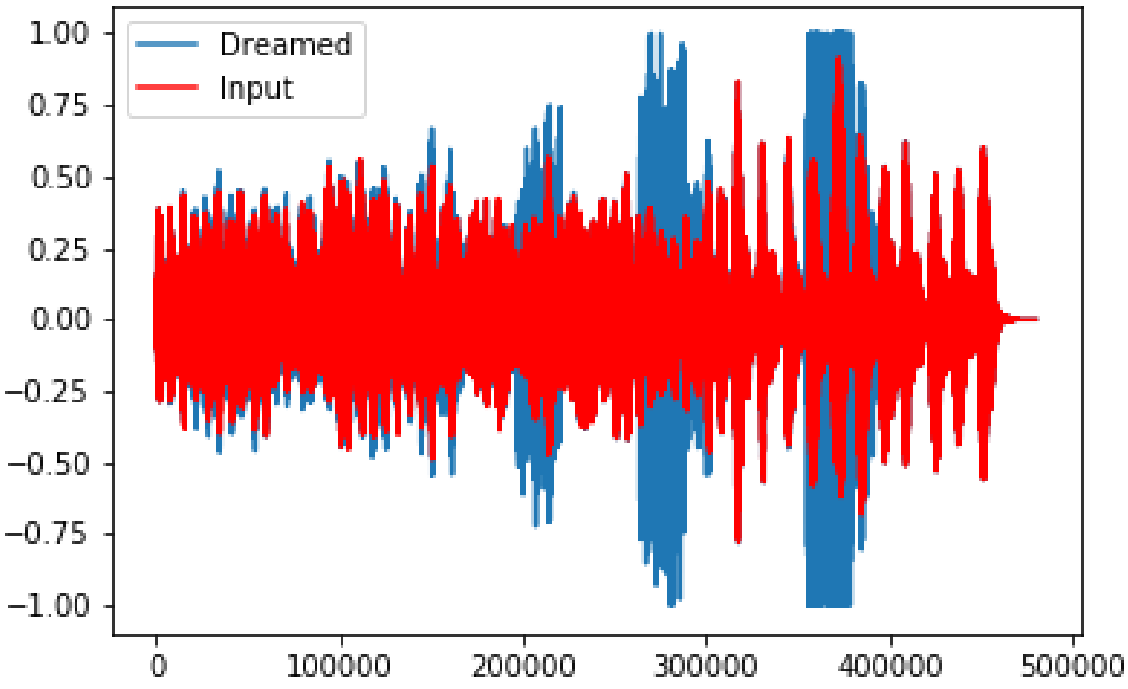

Figure A.50: Visualization of layer 4, filter 25

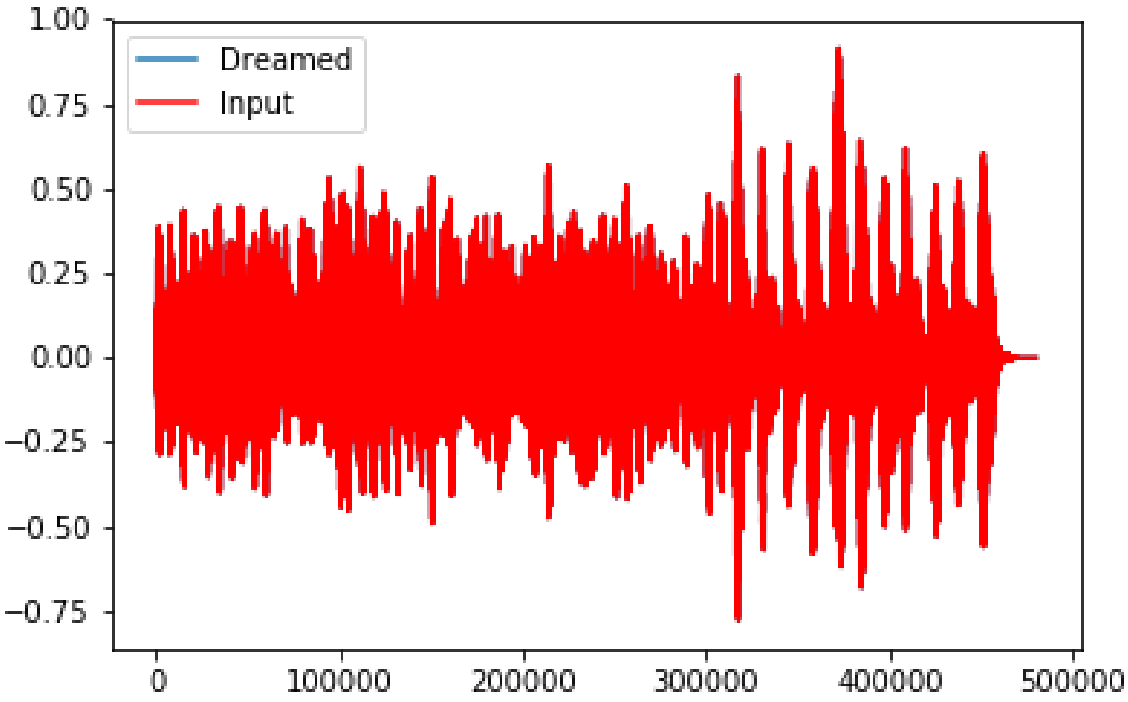

Figure A.51: Visualization of layer 4, filter 26

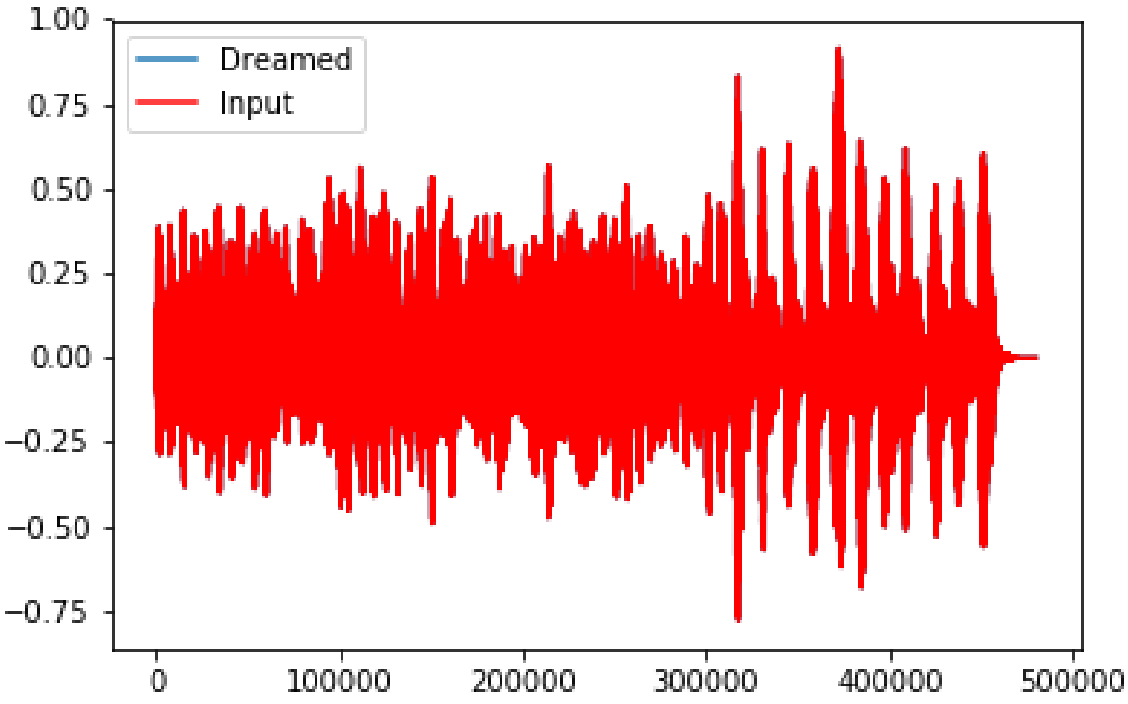

Figure A.52: Visualization of layer 4, filter 27

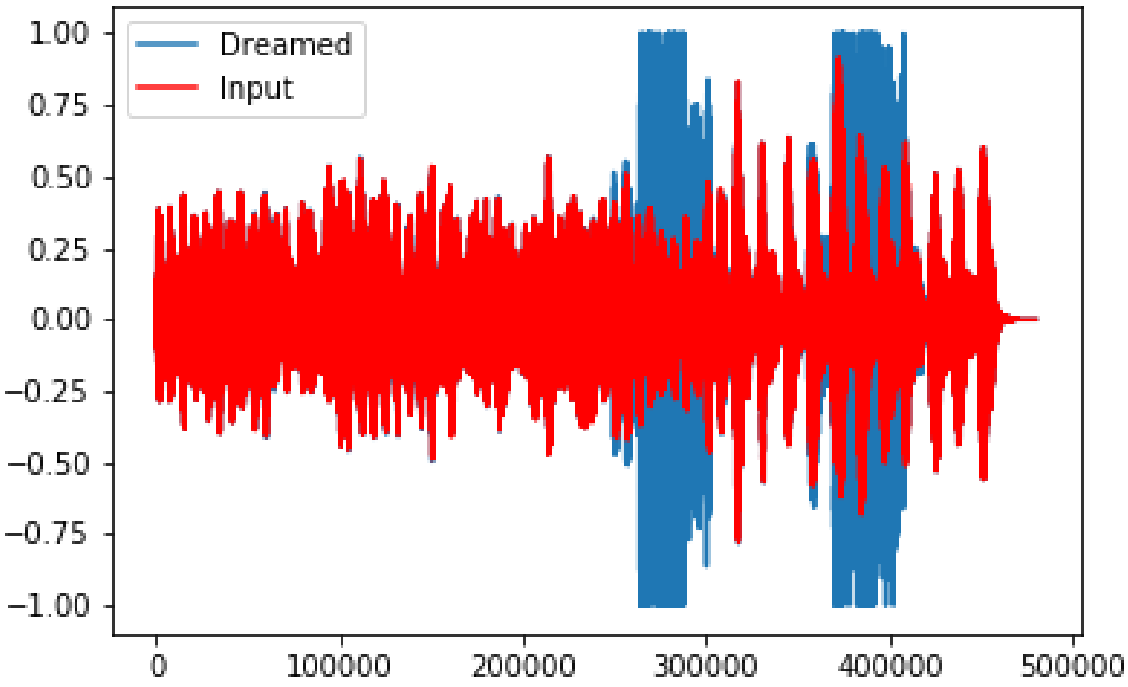

Figure A.53: Visualization of layer 4, filter 28

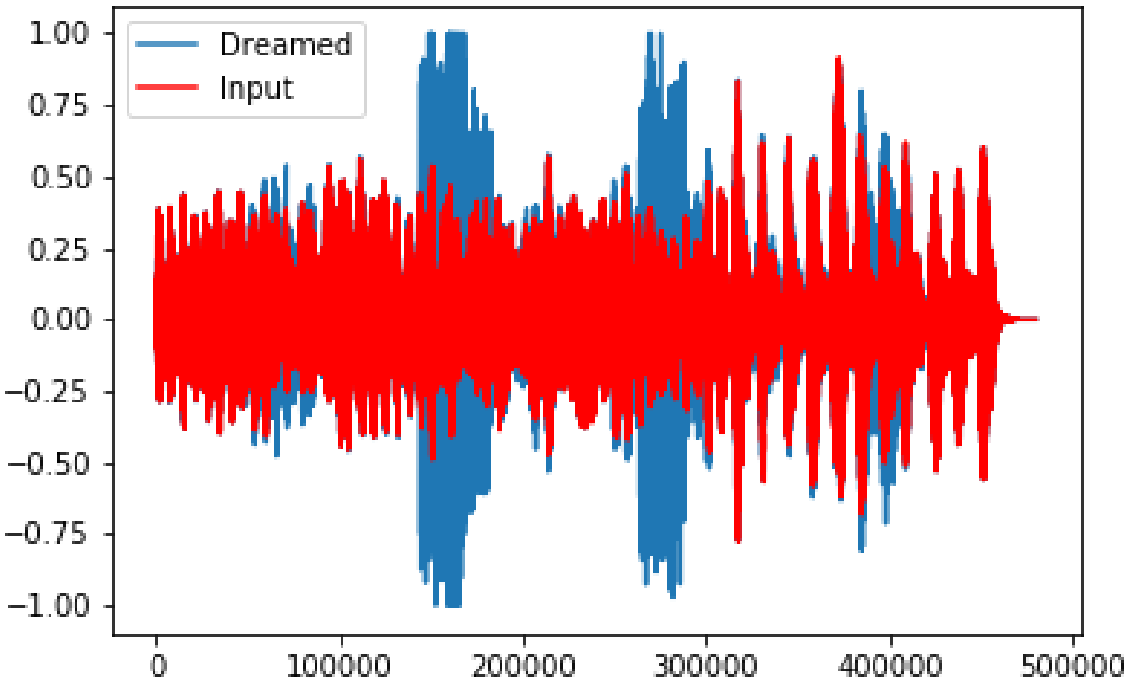

Figure A.54: Visualization of layer 4, filter 29

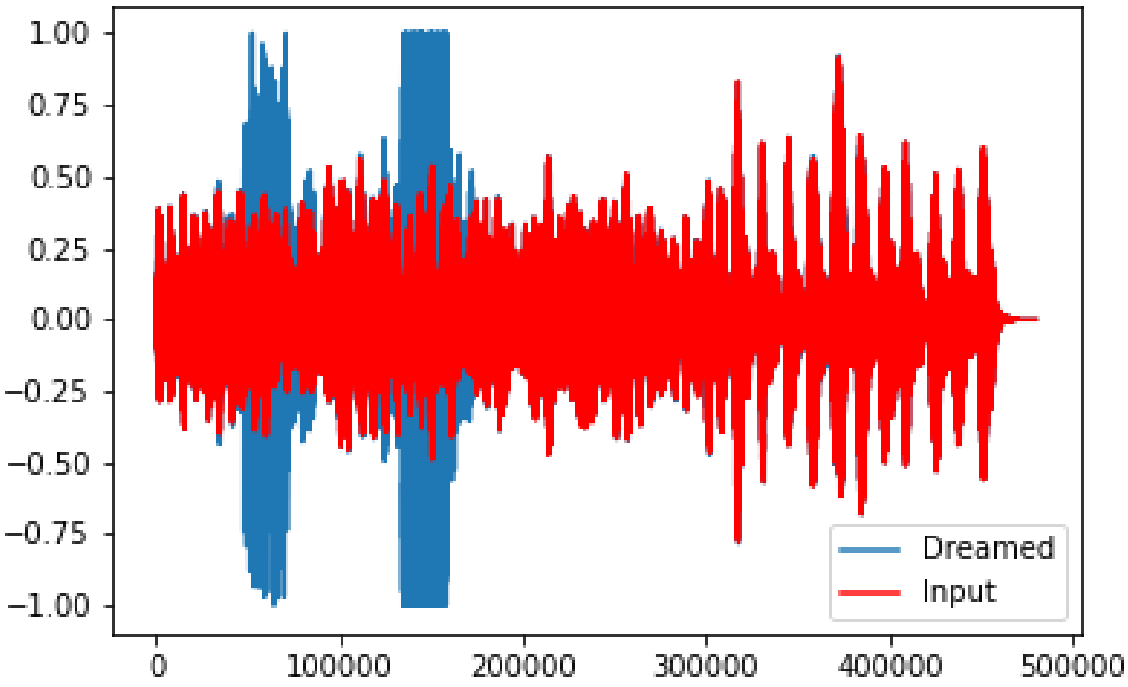

Figure A.55: Visualization of layer 4, filter 30

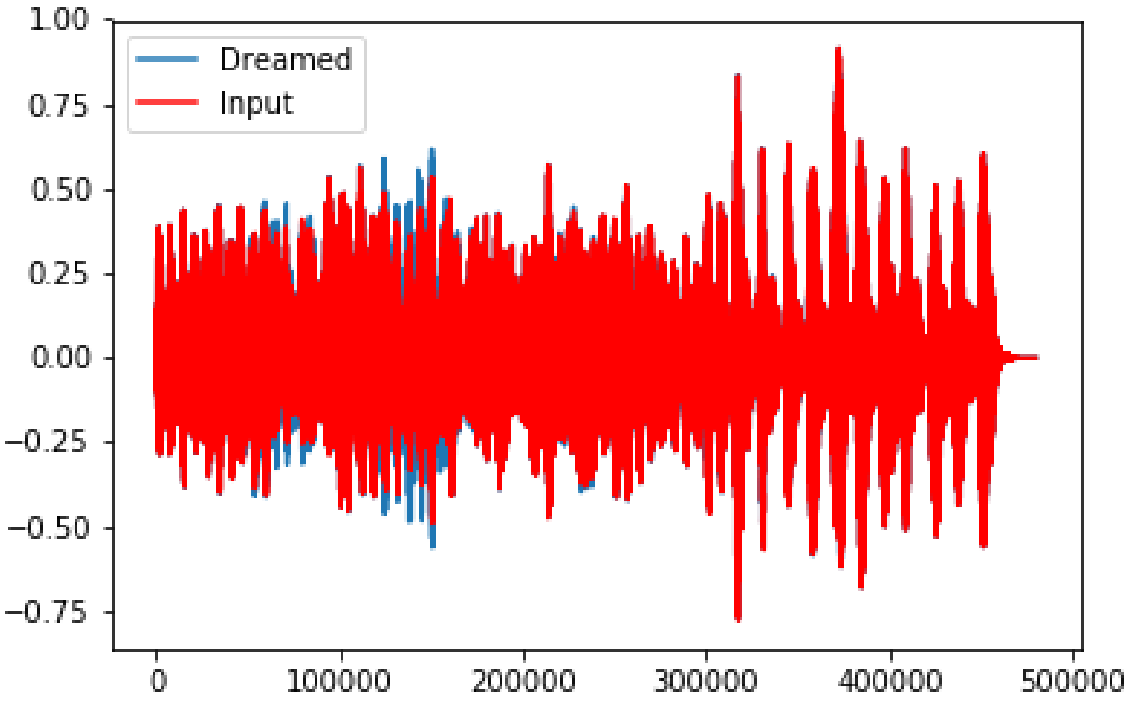

Figure A.56: Visualization of layer 4, filter 31