Visually Indicated Sound Effects

Joshua A. Baiad

May 2017

Department of Philosophy
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Mathematics
with an additional major in Logic & Computation.

To the Digirolamos, the Woolners, the Newmans, the Kents and every other family who has given me reason for hope and the strength to persevere.

Acknowledgments

I would like to thank Golan Levin for his guidance, without which I would never have been able to complete this work. I would also like to thank my reviewer and head of Carnegie Mellon University's Philosophy Department, David Danks, for giving me the freedom to pursue a thesis which enabled me to more deeply explore the intersection between the sciences and the arts.

Most of all, I would like to thank Andrew Hill, my high school math teacher and first true mentor. Without his guidance, I would have never pursued mathematics.

Contents

1

Introduction

There are a number of works which demonstrate that the auditory cortices of humans are capable of being stimulated by visual stimuli: deaf people "hear" sign language [14], we perceive phantom sounds when presented with sound-implying video [6, 12], and our perception of a phoneme's identity can be changed by a concurrent video of a mouth articulating a different phoneme (the McGurk effect) [11]. In some sense, the human brain contains a model capable of inferring audio from video. Can we develop a computational model which replicates this behavior with modern machine learning methods? In an attempt to develop a system which generates sound effects for silent videos, we examine this question through the lens of film and animation.

We take inspiration from [15], adopting much of their neural network architecture and computer vision techniques. While their work was limited to silent videos of people hitting and scratching objects with a drumstick, we attempt to examine a more dynamic corpus composed of episodes of the animated series *Tom & Jerry*. The more varied nature of this corpus requires our network to learn a target function with a much larger domain and range: videos from this corpus exhibit a greater range of actions and images, and the corresponding audio is significantly more complex. This suggests that our model must be capable of greater expressivity than what would be required to perform the same task on the corpus presented in [15].

We begin with a discussion of different methods one could use to model sound from video, propose our method, discuss our implementation, and finally evaluate our results with remarks about possible future work.

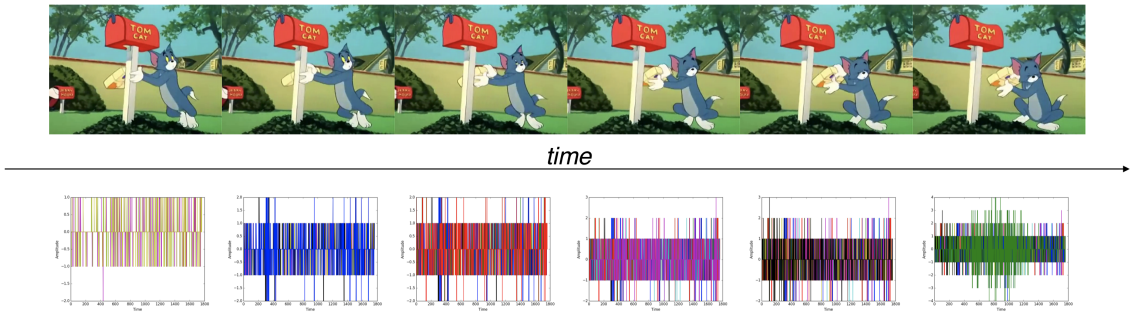
2

Modeling sound effects from silent video

2.1 Digital representations of video and audio

When we view film and listen to audio, we experience them in a way that seems continuous. However, both of these media are discrete in nature: digital video is nothing more than a sequence of images that are displayed at a frequency for which motion seems smooth to us, and digital audio is a sequence of discrete amplitudes which are transformed into pressure waves by headphones or speakers.

Figure 2.1: Video frames and their corresponding audio samples taken from an episode of *Tom & Jerry*



Then, the task at hand is to create a system which takes discrete video frames as input and outputs discrete amplitudes for the corresponding audio. More generally, we want to create a system which outputs a sequence Y when given some input sequence X. In our case, X is composed of video frames and Y is composed of amplitudes (i.e., X_i is the i-th video frame and Y_i is the i-th audio amplitude sample). In prior machine learning work, approaching this kind of sequential learning problem has been done with what we will call "sequence-agnostic" models (e.g., support vector machines, logistic regression, feedforward networks), Markov models, and recurrent neural networks (RNNs). In the following sections, we evaluate each of these architectures and discuss their suitability for our task.

2.2 Sequence-agnostic models

Support vector machines (SVMs), logistic regression, feedforward networks, and similar methods have demonstrated astounding results in many machine learning tasks, especially classification. We could frame our problem as a classification one: given a video frame, what is the audio clip that identifies or classifies it in some sense? Just like image labeling, this would be a classification problem where the input is of high dimensionality; however, unlike image classification, the output also has high dimensionality. Nonetheless, our task can be considered a classification problem, so it would not be unreasonable to apply techniques that have shown success in this area.

The problem with these models becomes clear when we consider the following: assume that the audio being generated for some frame X_i is dependent on the previous frame X_{i-1} and the audio that was generated for that frame. This is a reasonable assumption, as there are temporal dependencies across a piece of a music: these dependencies manifest themselves as musical themes, motifs, melodies, and other musical structures. Given the largely orchestral soundtrack for *Tom & Jerry* and its use of instrumentation to create sound effects, this assumption is also particularly fitting for our task. Ideally, then, we'd want our sequence-agnostic model to learn a mapping f which captures this notion of temporal or sequential dependency so that the audio generated for some frame X_i is different depending on what frame was just processed previously. The issue is that f is fixed, meaning that $f(X_i)$ will always be the same value regardless of what was previously input to or output by f—there is no dynamic behavior corresponding to the inherent sequential nature of our data. Thus, a mapping f learned by any sequence-agnostic model would be incapable of capturing dynamic behavior resulting from sequential dependencies across its input sequence.

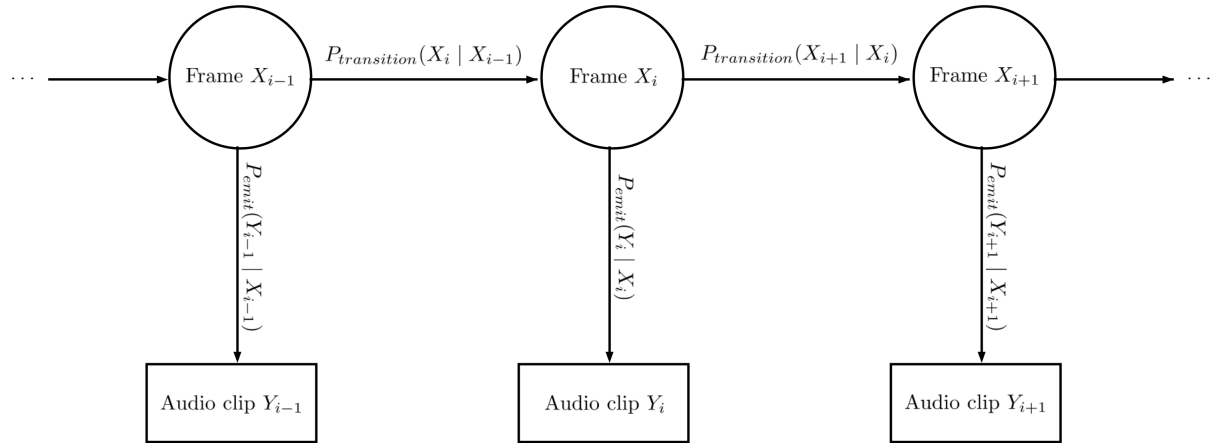
If models that fail to explicitly encode the sequentiality of the process being modeled aren't expressive enough for our purposes, then it is worth considering some models which do explicitly encode sequentiality.

2.3 Markov models

Markov chains, first described in 1906 by mathematician Andrey Markov, model sequential processes as a series of state transitions. Each element in the sequence can be viewed as a state, and the transition from one state to another is described by a probability distribution conditioned on the current state. Hidden Markov models (HMMs) extend this idea: the sequential process being modeled has observable and unobservable (hidden) states, and both the current hidden state and next observable state are modeled with probability distributions conditioned on the current observable state.

Markov models like these make the strong assumption that the underlying process is Markovian in nature: the idea that the next state in the sequence is dependent only upon the current state. This assumption prevents Markov models from modeling long-range dependencies in the sequence where, for example, the presence of a state in the sequence is strongly affected by a state many steps back in the sequence. Since music and video often have long-ranging temporal dependencies, like the one noted in our example, Markov models arent suitable for our modeling

Figure 2.2: Hidden Markov model formulation of our sequence learning task



task.

Furthermore, when the state space is large (as it is in video and audio), performing inference with HMMs becomes computationally impractical. If we have a state space S, the dynamic programming algorithm to perform inference (the Viterbi algorithm) has $O(|S|^2)$ complexity. Additionally, if we have a state space S, then the transition matrix which contains the probabilities which describe how states move from one to another is of size $|S|^2$. Thus, for videos with even relatively small height, width, and channel dimension, the standard inference operations for HMMs become intractable. Moreover, the performance of these operations become even worse if we attempt to look more than one state into the past in an attempt to capture the long-term state dependencies which are beyond the reach of standard Markov models.

Even without these performance concerns, if we changed our Markov model to look more than one state into the past when evaluating the current state, we must decide how far we look back at the time of inference. This forces us to make *a priori* assumptions about our data. We could, theoretically, circumvent this by looking back to the initial state every time we perform inference for a new state, but as our state sequence grows in size, we run into memory limitations. If we would like to process state sequences of arbitrary length, this method of capturing sequential dependencies is not practical.

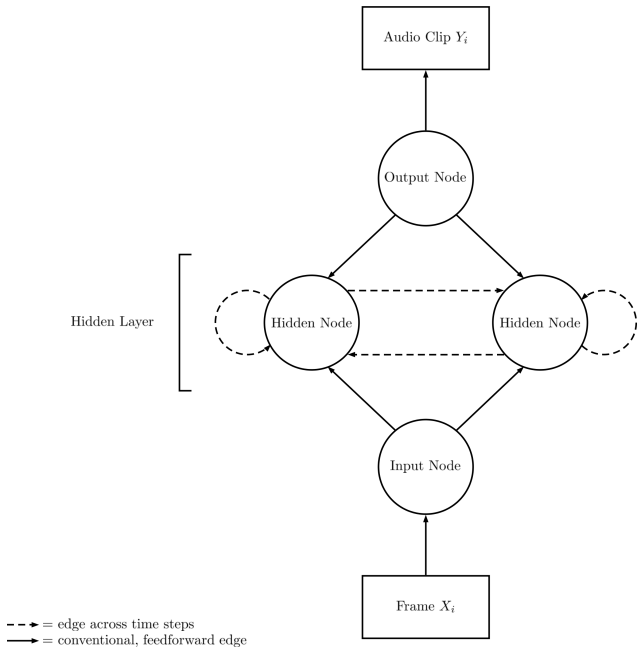
2.4 Context windows

Similar to Markov models, it is possible to change our sequence-agnostic models in a way which enables them to look into previous or future states up to a certain depth. We do this by providing sequence-agnostic models with a context window around the current frame being processed as a means of implicitly capturing temporal/sequential dependencies both in the future and the past. That is, we could change our model to learn a mapping f so that we compute something like $f(X_{i-1}, X_i, X_{i+1})$ instead of $f(X_i)$, where X_i is the frame we're currently processing to generate audio. This approach has been used before with deep neural networks (DNNs) for the purposes of speech modeling [9]; however, even with this approach, we are ultimately limited

by the size of our sliding context window: if our context window contains information about the previous p frames and the next n frames, we'll never be able to train our model to understand temporal dependencies which extend beyond p frames in the past or n frames in the future. Without extensive domain knowledge and preprocessing, we know nothing about the length of the temporal dependencies in a given piece of video or audio. Consequently, without making *a priori* assumptions about our data, we could never be certain that a sequence-agnostic model would fully capture every temporal dependency. For the sake of generalizability, it is necessary that our model has the expressiveness to capture temporal dependencies of arbitrary length.

2.5 Recurrent neural networks

Figure 2.3: RNN formulation of our sequence learning task



In 1991, Hava Siegelmann and Eduardo Sontag proved that finitely sized recurrent neural networks with only linear transfer functions are just as computationally powerful as Turing machines [20]. The ideas presented in their work were furthered by Heikki Hyötyniemi's 1996 paper, *Turing Machines are Recurrent Neural Networks* [5]. Their results demonstrate that finitely-sized RNNs are capable of learning *any* computable function. In any case, then, RNNs overcome the expressivity challenges encountered by sequence-agnostic models. Furthermore, as demonstrated in [20], this expressivity is captured by RNNs of finite size, which means that we avoid the impractical memory requirements of Markov models with similar expressivity. To understand RNNs, we first need to discuss neural networks.

In general, a neural network consists of a set of artificial neurons, commonly referred to as nodes or units, and a set of directed edges between them. Intuitively, these edges correspond to

the synapses in a biological neural network, like those which make up the human brain. Feedforward networks are neural networks in which there are no cycles. RNNs are similar in that there are no cycles among conventional edges, but allow cycles across time-steps. These cycles over time allow the neural network to maintain some notion of memory while evaluating a sequence of inputs. For this reason, we can say that RNNs are "stateful" in some sense, while feedforward networks are not.

The general architecture of RNNs will be the bedrock upon which our model learns the sequential nature of videos and audio.

2.6 Training recurrent neural networks

The traditional method of training a neural network is via gradient descent. Generally, this is implemented as the backpropagation algorithm, a fairly straightforward application of the chain rule from multidimensional calculus. In the case of RNNs, we use a variant of backpropagation called backpropagation through time (BPTT). Rather than only considering the order of node computation as in standard backpropagation, BPTT requires us to also consider the order of the recurrent computations when evaluating ordered partial derivatives. For a complete discussion of backpropagation, BPTT, and the underlying time scales calculus upon which they are built, we refer the reader to [16, 19, 22].

Training recurrent neural networks is not simple, however, as the phenomenon of vanishing and exploding gradients can result in problematic behavior. As explained by Pascanu, Mikolov, & Bengio:

> "the *exploding gradients* problem refers to the large increase in the norm of the gradient during training. Such events are due to the explosion of the long term components, which can grow exponentially more than the short term ones. The *vanishing gradients* problem refers to the opposite behavior, when the long term components go exponentially fast to norm 0, making it impossible for the model to learn correlation between temporally distant events" [16].

At a very high level of abstraction, we can view the exploding gradient problem as an instance where the network puts too much weight on past inputs and outputs, causing short-term dependencies to be ignored. Similarly, the issue of vanishing gradients occurs when the network puts too much weight on short-term dependencies. We'd like to avoid both of these scenarios so that the network learns when to look far into the past and when to look at only temporally recent or local events. One way to avoid this issue is to perform truncated backpropagation, which effectively imposes a limit on how many time steps into the past our network examines when calculating the gradient. However, using this method is similar to the sliding context window from our discussion on sequence-agnostic models: it presupposes the maximum temporal distance across which dependencies exist within our input and output sequences.

Fortunately, there exist recurrent neural network architectures for which exploding and vanishing gradients is not a concern.

2.7 Long short-term memory

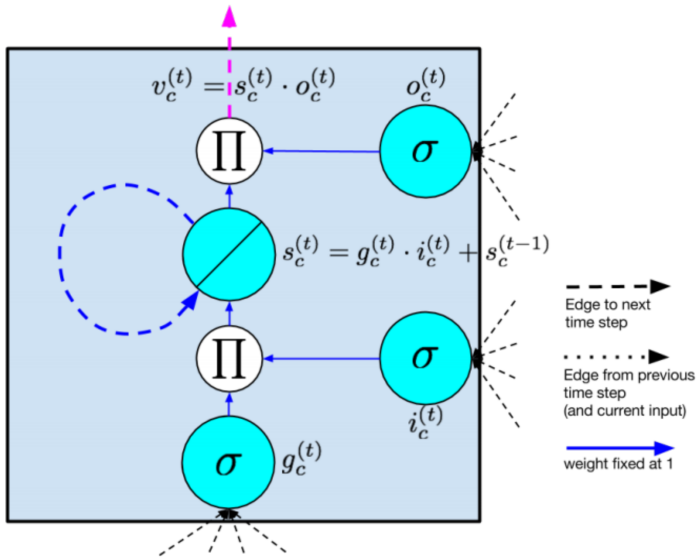


Figure 2.4: A diagram of the memory cell of an LSTM unit taken from [7]

In *Long Short-Term Memory*, Hochreiter and Schmidhuber introduce the idea of a memory cell, which replaces the standard neurons in the hidden layer of the RNN architecture [4]. The memory cell is a unit designed for the purpose of overcoming the exploding and vanishing gradient problems. A memory cell consists of the following components:

- **Input node**: This component applies an activation function (typically *tanh* or a *sigmoid*) to the weighted sum of the input at the current time, plus the weighted sum of the output from the hidden layer at the previous time step, plus a bias vector.
- **Input gate**: Like the input node, the input gate takes input from the current time and from the hidden layer at the previous time step and applies an activation function. The resulting vector is then multiplied against the input node and fed to the internal state. It is for this reason that we use the term "gate", as the vector fed to the internal state will be 0 if the gate's value is 0 and, similarly, the vector fed to the internal state will be the value from the input node if the gate's value is 1. Thus, the input gate controls the flow from the input node to the internal state of the memory cell.
- **Internal state**: This is the component by which the memory cell circumvents the exploding and vanishing gradient problem. The internal state serves as some form of intermediate storage across time steps through a self-connected recurrent edge. This recurrent edge is fixed with a weight of one to ensure that the gradient can pass across many time steps without vanishing or exploding.
- **Forget gate**: Although not included in the original specification outlined in [4], most modern implementations of long short-term memory (LSTM) units include this component. The forget gate is used as a mechanism for the memory cell to flush the contents of its internal state.

- **Output gate**: Similar to the other gate components, the output gate controls the flow out of the memory cell.

When we train our LSTM units, the network learns how to allow activation values into and out of memory cells—in some sense, LSTM units learn when to let error in, and when to propagate error out. It is with these LSTM units that we will construct our RNN and avoid the training issues presented by vanishing and exploding gradients, thereby enabling us to capture arbitrarily long temporal dependencies in our input and output sequences.

2.8 Bidirectional recurrent neural networks

Previously, we noted that it is a reasonable assumption that the audio currently being generated depends on audio that was generated in previous time steps. However, it is also reasonable to believe that the audio currently being generated depends on audio that will be generated in future time steps. For example, if the current audio being generated corresponds to some melody, the current pitch being generated must fall in line with future pitches in the melody. While we are not generating audio pitch-by-pitch, the main idea still applies. To fully capture both dependencies in the future and the past, we would like to use an RNN architecture that can look into the future as well as the past.

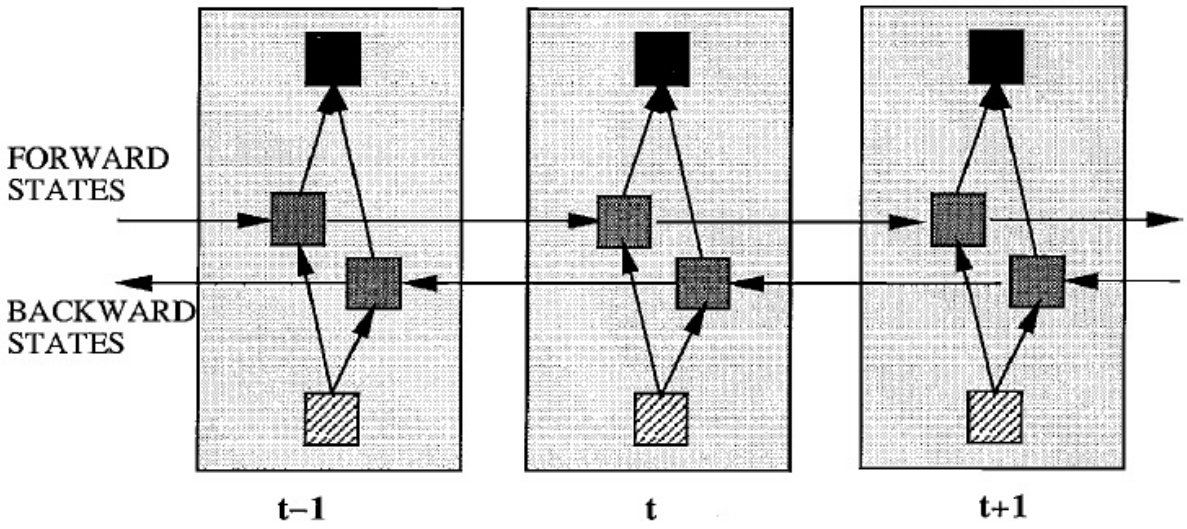


Figure 2.5: Diagram of BRNN architecture taken from [18]

In their 1997 paper, Schuster and Paliwal outline an architecture for RNNs that looks both forwards and backwards in time [18]. The unique feature about the bidirectional recurrent neural network (BRNN) architecture is that there are two hidden layers, both of which are connected to the input and output layers. The two hidden layers are differentiated by their recurrent connections: one layer has recurrent connections *from* past time steps while the other layer has recurrent connections *to* past time steps, passing activation values backwards along the sequence.

Like other RNN architectures, BRRNs can be trained with gradient descent methods like backpropagation. Additionally, BRNNs can be integrated with LSTMs, allowing us to capture temporal dependencies of arbitrary length that extend both far into the past or future. However, one major limitation of BRNNs is that they cannot be used for real-time computation, as it requires datapoints from the future. It is possible, though, to use BRNNS for buffered computation, depending on how much latency we are willing to tolerate.

To fully capture temporal dependencies both in the future and the past of our input and output sequences, we propose a BRNN architecture with LSTM units. Now, it remains to provide a more rigorous specification for how we will represent audio and video within our model.

3

Feature extraction

3.1 Computer vision

If our goal is to generate sound effects for silent video, then it is worth noticing that most sound effects, especially those in an animation like *Tom & Jerry*, correspond to onscreen motion. Then, we'd like to attempt a number of techniques which detect and isolate motion from the rest of the video so that our model can more easily learn the association between motion and audio. However, if we are also concerned with generating additional audio, like the soundtrack and other background noise, such motion detection techniques could hinder rather than aid our model.

3.1.1 Motion Detection

Frame differencing

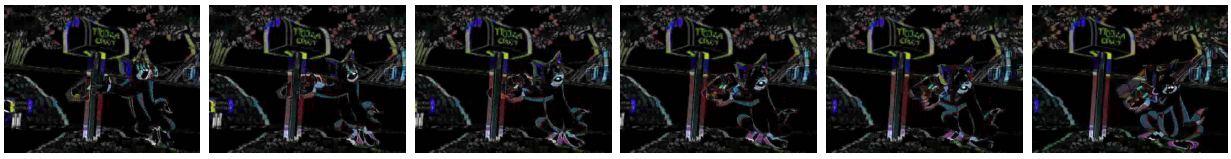


Figure 3.1: Example of calculating the frame differences between frames in an episode of *Tom & Jerry*

A common and straightforward way of detecting motion is to calculate the frame difference between adjacent frames. We compute this by performing element-wise subtraction along an image's pixels. So, if I and I' are two images, we compute the pixel for the frame difference between I and I' at coordinates (i, j) as follows:

$$I_{\text{diff}}[i, j] = I[i, j] - I'[i, j].$$

Thus, if we were to calculate the frame differences for a sequence of frames in which nothing is moving, the resulting video would be a constant black image. However, if the sequence of frames corresponded to a video in which an object is moving, unmoving elements of the frame, such as background scenery, would be black and the moving object would remain visible. By calculating the difference between adjacent frames we are, in some sense, "subtracting" the background.

Optical flow

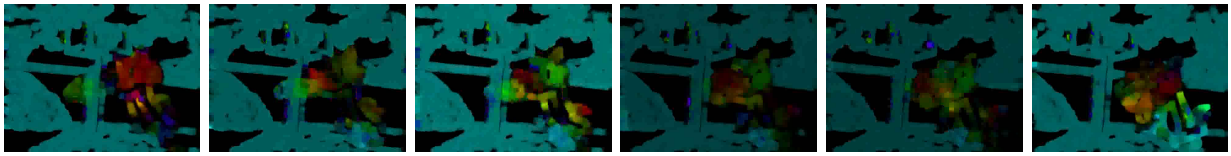


Figure 3.2: Optical flow across 6 frames of an episode of *Tom & Jerry* using the method introduced in [3]

One of the limitations of frame differencing is that it doesn't provide any information about the direction or magnitude of the motion. Surely the sound for a fast moving object is different than a slowly moving one, but computing the frame difference fails to explicitly capture that visual distinction. According to Burton and Radford [17], the term "optical flow" is defined as: "the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene." In their analysis of various optical flow algorithms, Kondermann et. al argue that "the aim of optical flow (OF) algorithms is to compute a motion vector field based on an image sequence". Thus, optical flow algorithms are capable of determining the direction and magnitude of movements seen in a video which are not explicitly captured by an approach which uses frame differences.

Spacetime images

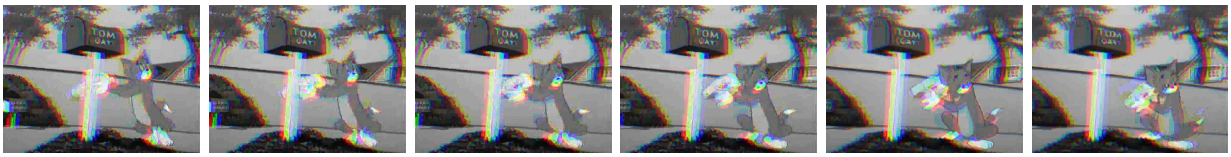


Figure 3.3: Spacetime images for 6 frames of an episode of *Tom & Jerry*

Another method of preprocessing videos in such a way that highlights motion is to generate *spacetime images*, a technique introduced in [15]. The spacetime image S_t for frame t is the image whose three channels are grayscale versions of the previous, current, and next frames. That is, S_t is defined by

$$S_t[i, j] = (\text{Grayscale}(F_{t-1}), \text{Grayscale}(F_t), \text{Grayscale}(F_{t+1})),$$

where F_t is the t-th frame. This representation encodes motion across video frames by providing a context window around the frame of interest, similar to the context windows we've discussed earlier. Like optical flow images, spacetime images capture some notion of a temporal derivative.

In our implementation, we train four different models with different video preprocessing procedures: one model is trained on the raw video, and three other models are trained on video that has been preprocessed with one of the above methods of motion detection.

3.1.2 Convolutional neural networks

It is useful to note that raw images are stored as multidimensional arrays in memory. Black-and-white images are stored as two-dimensional arrays I, where $I_{i,j}$ is the pixel at coordinates (i, j) whose value is generally ranges between 0 (corresponding to black) and 255 (corresponding to white). For most colored images, $I_{i,j}$ is commonly a triple (r, g, b) where r corresponds to the red intensity, g corresponds to the green intensity, b corresponds to the blue intensity of the image, and each value ranges from 0 to 255. Then, a colored image that is h pixels tall and w pixels wide, where each pixel is represented by three 8-bit integers (the fewest number of bits needed to encode a value ranging from 0 to 255), requires at least $24hw$ bits or, equivalently, $3hw$ bytes. If we're dealing with many images at once, as is the case in video, it may be easier and faster to deal with some alternative, lower-dimensional representations.

One way to go about this is to perform feature extraction on each video frame. The question, then, is what features do we extract? By deciding upon an explicit set of hand-crafted features to extract, we implicitly make an assumption about what image features are relevant for the purposes of sound generation. Since we prefer to avoid *a priori* assumptions about our data and the underlying relationship between the video and the audio for the sake of generalizability, it would be better to find a way to *learn* the relevant features. This can be done with convolutional neural networks (CNNs), which effectively learn a filter to apply to an image. CNNs are used for this purpose on spacetime images in [15]. Additionally, CNNs have been used for many image classification tasks which require feature extraction, such as video description [2] and action recogntion [21]. Like the other neural network architectures we've discussed, CNNs can be trained by a gradient descent implementation like backpropagation.

At the core of CNNs is the convolutional layer. This layer performs what is known as the convolution operation: it takes its input and convolves it with a convolution kernel. This is perhaps made most clear by the example given in Figure 3.4.

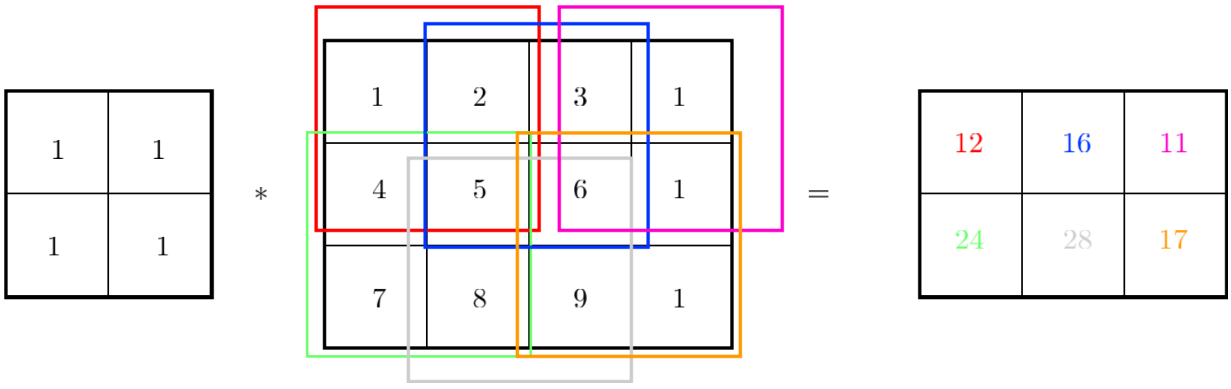


Figure 3.4: Convolving a 2×2 kernel with a 3×4 input matrix with a stride of size 1

Essentially, we perform convolution by taking the convolution kernel, overlapping it with the input matrix and performing element-wise multiplication between the numbers in the kernel and the matrix where the overlap occurs. Then, we sum the results of these multiplications to a single number. During training, the convolutional layer learns what values should be held in the cells

of the convolution kernel. In this way, the convolutional layer can learn how to "reduce" regions of pixels into a single scalar feature.

3.2 Audio feature extraction

Attempting to generate arbitrarily long sequences of amplitudes is computationally difficult for one reason: the state space is enormous. For a sequence of amplitudes of length n, the state space is $\mathbb{R}^n$. Thus, our state space grows exponentially with respect to the length of the sequence we're attempting to generate. Our task would be made easier by shrinking the state space. One way to accomplish this is by computing audio features rather than raw audio amplitudes. By computing audio features, we can have a single fixed-length vector of scalar values that correspond to many thousands of amplitudes. Then, from these audio features, we can perform parametric inversion or use the vector as a key into a waveform database to retrieve the corresponding audio. This technique is used in [15] and in a number of other works on generative audio, largely due to the fact that generating raw audio is a difficult task without enormous neural networks and the necessary hardware to train them.

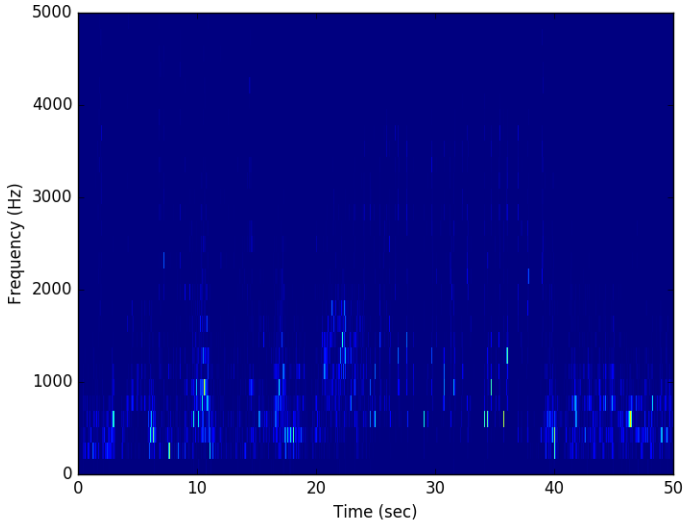
We will provide a brief overview of various audio feature extraction techniques but avoid going into the signal processing theory which motivates them, as that would warrant another paper in and of itself.

3.2.1 Mel-frequency cepstral coefficients

Mel-frequency cepstral coefficients (MFCCs) have shown great promise in speech recognition, music modeling, and speaker identification [1, 8, 10, 13]. MFCCs are computed by the following procedure:

1. Divide the audio signal into frames, usually consisting of 20ms worth of audio samples.
2. Take the discrete Fourier transform of each frame.
3. Calculate the logarithm of the amplitude spectrum, discarding phase information. We take the logarithm of the amplitude spectrum because the perceived loudness of a signal has been found to be approximately logarithmic in psychoacoustic studies.
4. Next, we perform Mel-scaling and smoothing, emphasizing perceptually meaningful frequencies. We do this by collecting spectral components and dividing them into frequency bins. These frequency bins are not equally spaced, as it has been found that lower frequencies are more important than higher frequencies, especially in the case of speech.
5. Finally, we perform a discrete cosine transform. As noted in [8], "the Mel-spectral vectors calculated for each frame are highly correlated. Speech features are typically modeled by mixtures of Gaussian densities. Therefore, in order to reduce the number of parameters in the system, the last step of MFCC feature construction is to apply a transform to the Mel-spectral vectors which decorrelates their components." The discrete cosine transform provides a way of approximating this decorrelation procedure and is used frequently within the speech community.

3.2.2 Spectrograms



Spectrograms provide a way to take a sound, decompose it into its various constituent frequencies through a Fourier transform, and determine the amplitude of each of these frequency. The amplitude of a constituent frequency is indicative of to what extent that frequency contributes to the original sound. Thus, for a given sequence of audio samples, we can perform a Fourier transform and place the frequencies into bins to yield a spectrogram which reduces the original audio into a much smaller vector of amplitudes.

3.2.3 Cochleograms

Cochleograms are much like spectrograms in that they provide a representation of the time-frequency plane. However, cochleograms are different in that they are based on excitation of the basilar membrane inside the cochlea. For this reason, it might be argued that cochleograms are better representations of how sound is perceived and processed by humans. Cochleograms are used as a form of audio features in [15] which are later parametrically inverted to waveforms.

Despite the success seen with MFCCs, the process of going from audio to MFCC loses a lot of information about the audio's frequency spectrum. Thus, inversion methods for MFCCs result in sound of very poor quality, though intelligible. Since we are generating audio from our features, this is undesirable. On the other hand, both cochleograms and spectrograms offer relatively easy inversion. Given the success of cochleograms in [15] and their more biologically informed design, we will use them as audio features in our model.

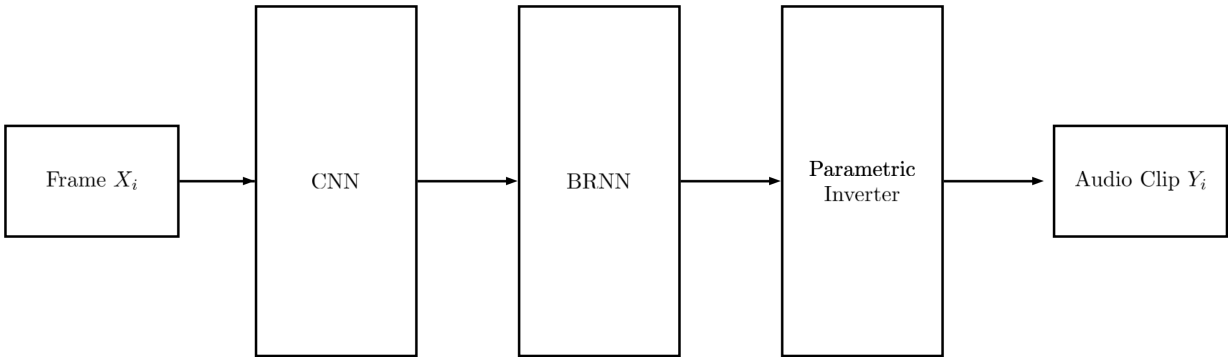
4

Methodology

Now that we have laid the necessary groundwork, we can provide a full specification of our model and procedure.

4.1 Model architecture

We propose a CNN-BRNN architecture where the BRNN consists of LSTM units. This network will receive video frames as input and output cochleograms which are then fed through a parametric inverter to yield corresponding waveforms:



4.2 Corpus

Our corpus consists of 161 epsiodes of *Tom & Jerry*, each approximately 7 minutes long for a total of approximately 19 hours of video and audio data. The episodes range across several decades, so there are slight variations in animation style and video/audio quality. This introduction of variability may impose enough noise in our data that limits significant overfitting, but we have yet to verify this in any way. We train on 70% of the corpus and test against the remaining 30% of the episodes.

4.3 Implementation

Prior to building and training the model, we preprocessed each of our videos and their corresponding audio to fit the input/output schemata we discussed in the previous section. That is, we processed each video to generate their frame differences, optical flow, and spacetime images. Then, we processed the audio for each video to generate their cochleograms. Finally, with the TensorFlow framework, we trained four separate models with our proposed architecture, cochleograms as output, and the following as input: raw video, frame differences, optical flow, and spacetime images. Additionally, we use the $L2$ norm as our loss function with the reasoning that small deviations in our model's output from the target output should not be penalized as highly as large deviations: a slight difference in frequency or amplitude is not as noticeable as a large difference, and this is reflected in the way humans perceive audio.

5

Conclusion

5.1 Results

We performed several hundred epochs of training iterations across two Nvidia GeForce GTX 780s. The audio generated by our model on the testing set of episodes exhibits behavior that seem to be correlated with visual events, leading us to believe that the model does not lack the necessary expressivity to learn the audio-visual mapping. However, at the time of writing, current results indicate that more training time is necessary. The audio resulting from our model sounds faint and muddled behind a fair degree of noise, suggesting that our model would benefit from many more epochs of training. Due to this, it is also difficult to compare the audio generated by our models trained on different video input at this time.

Nonetheless, the presence of visually indicated sound in our generated audio is promising. If further training yields less noisy audio with even stronger correlation to onscreen events, our model may provide the groundwork for automatic film scoring and automatically generated sound effects similar to those produced by Foley artists.

5.2 Future work

In our discussion of sequence-agnostic models, we noted how our task could be posed as a classification problem. However, evaluating our model's performance is not as straightforward as it is for classification tasks. It is impossible to provide a meaningful notion of classification rate or accuracy for our problem: just because the audio generated by our model may not be close (at least with respect to our loss function) to the audio which we test it against does not necessarily mean it does not "fit" the video. If we were to evaluate our model as we would in a classification setting, we would need a way to measure how well a given piece of audio "fits" a video. Only with this would we be able to create a meaningful notion of accuracy for our model.

Realistically, this would require human trials in which participants tag audio-video pairs as "fitting" or "not fitting". A participant would be given a video to watch alongside either the original audio or the audio generated by our model. Then, the participant would provide one of the two labels for the audio-video pairing they observed. The difference between the rate at which participants classify an audio-video pair with audio generated by our model as "fitting"

and the rate at which they classify an audio-video pair with the video's original audio would give us some idea of how accurate our model is, at least with respect to audio composed and generated by humans.

Bibliography

[1] P. M. Chauhan and N. P. Desai. Mel frequency cepstral coefficients (mfcc) based speaker identification in noisy environment using wiener filter. In *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*, pages 1–5, March 2014. doi: 10.1109/ICGCCEE.2014.6921394. 3.2.1

[2] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014. URL `http://arxiv.org/abs/1411.4389`. 3.1.2

[3] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40601-8. URL `http://dl.acm.org/citation.cfm?id=1763974.1764031`. 3.2

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`. 2.7

[5] Heikki Hyotyniemi. Turing machines are recurrent neural networks, 1996. 2.5

[6] P. j. Hsieh, J. T. Colas, N. Kanwisher, P. j. Hsieh, J. T. Colas, and N. Kanwisher. Spatial pattern of bold fmri activation reveals cross-modal information in auditory cortex, 2012. 1

[7] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. URL `http://arxiv.org/abs/1506.00019`. 2.4

[8] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *In International Symposium on Music Information Retrieval*, 2000. 3.2.1, 5

[9] Andrew L. Maas, Quoc V. Le, Tyler M. Oneil, Oriol Vinyals, Patrick Nguyen, and Andrew Y. Ng. Recurrent neural networks for noise reduction in robust asr. 2.4

[10] J. Martinez, H. Perez, E. Escamilla, and M. M. Suzuki. Speaker recognition using mel frequency cepstral coefficients (mfcc) and vector quantization (vq) techniques. In *CONIELECOMP 2012, 22nd International Conference on Electrical Communications and Computers*, pages 248–251, Feb 2012. doi: 10.1109/CONIELECOMP.2012.6189918. 3.2.1

[11] Harry McGurk and John MacDonald. Hearing lips and seeing voices. *Nature*, 264:746–748, 12 1976. 1

[12] Kaspar Meyer, Jonas T. Kaplan, Ryan Essex, Cecelia Webber, Hanna Damasio, and Antonio

Damasio. Predicting visual stimuli on the basis of activity in auditory cortices. *Nature Neuroscience*, 13(6):667–668, May 2010. ISSN 1097-6256. doi: 10.1038/nn.2533. URL http://dx.doi.org/10.1038/nn.2533. 1

[13] Lindasalwa Muda, Mumtaj Begam, and I. Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *CoRR*, abs/1003.4083, 2010. URL http://arxiv.org/abs/1003.4083. 3.2.1

[14] H. Nishimura, K. Hashikawa, and T. Kubo. Sign language 'heard' in the auditory cortex. *Nature*, 397:116, 1999. 1

[15] Andrew Owens, Phillip Isola, Josh H. McDermott, Antonio Torralba, Edward H. Adelson, and William T. Freeman. Visually indicated sounds. *CoRR*, abs/1512.08512, 2015. URL http://arxiv.org/abs/1512.08512. 1, 3.1.1, 3.1.2, 3.2, 3.2.3

[16] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063. 2.6

[17] John. Radford and Andrew. Burton. *Thinking in perspective : critical essays in the study of thought processes / edited by Andrew Burton and John Radford*. Methuen London, 1978. ISBN 0416858406 0416858309. 3.1.1

[18] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997. ISSN 1053-587X. doi: 10.1109/78.650093. URL http://dx.doi.org/10.1109/78.650093. 2.5, 2.8

[19] J. Seiffertt and D. C. Wunsch. Backpropagation and ordered derivatives in the time scales calculus. *IEEE Transactions on Neural Networks*, 21(8):1262–1269, Aug 2010. ISSN 1045-9227. doi: 10.1109/TNN.2010.2050332. 2.6

[20] Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4:77–80, 1991. 2.5

[21] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014. URL http://arxiv.org/abs/1406.2199. 3.1.2

[22] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. Reprinted in [**?**]. 2.6