RECONSTRUCTING SPEECH INPUT FROM CONVOLUTIONAL NEURAL NETWORK ACTIVITY

Bachelor's Thesis in Artificial Intelligence

Thesis Submitted by

Thomas Churchman

s4206606

Under Supervision of

Marcel van Gerven[‡]

and

Umut Güçlü[‡]

[‡] Donders Institute for Brain, Cognition and Behaviour

In Partial Fulfillment for the Degree of

B.Sc. (Artificial Intelligence)

July 08, 2015

Department of Psychology and Artificial Intelligence

Radboud University Nijmegen

Abstract

Convolutional Neural Networks (CNNs) applied to the auditory domain have achieved great results. However, little research has been performed to uncover the underlying mechanisms that allow auditory CNNs to reach these achievements. This exploratory research attempts to help uncover these mechanisms by using a CNN's activation patterns generated by speech inputs to reconstruct those inputs. It is found that training a decoder to decode activity patterns to a preliminary reconstruction and consequently fine-tuning that reconstruction through further backpropgation generates the best results. The reconstructions show that the network preserves a good representation of the input up to and including the fully connected units. Reinforcingly, this representation appears to be suited to speech, instead of, for example, audio in general. Furthermore, it becomes apparent that the network is insensitive to input intensity as well as to the input's activation scale on the time-domain. Further research is required to discover more properties of CNNs applied to the auditory domain.

1 Introduction

Convolutional Neural Networks (CNNs) are extremely adept at encoding and recognizing objects in the visual domain. For example, Krizhevsky, Sutskever, and Hinton (2012) trained a CNN on an ImageNet dataset consisting of 1.2 million images and 1,000 classes. They achieved a classification error rate of 17%, which was better than the previous state of the art. The great success of these networks on the visual domain has made researchers eager to achieve similar success with these networks applied to the auditory domain. This research has been met with success as well. For example, Abdel-hamid, Jiang, and Penn (2012) created a phone recognition model utilizing CNNs that overtook previously attained results in similar experimental setups.

In spite of the success achieved by CNNs on the auditory domain, and whereas the internal representations of CNNs in the visual domain have been extensively investigated (e.g., M. D. Zeiler and Fergus (2014)), only limited such research has been performed for CNNs on the auditory domain (e.g., Ma, Dang, and Li (2014)).

The main goal of this research is to help uncover the underlying mechanisms that CNNs on the auditory domain employ to reach their achievements. To this end, it is interesting to look at the possibility of reconstructing speech input from the activation patterns generated by this input. It is expected that such a reconstruction can be generated if the CNN truly creates a good internal representation of the input.

If reconstruction can be performed successfully, specific patterns found across reconstructions can be investigated. These patterns might shine light on the internal representations of the input the network employs. Having a deeper understanding of these representations will help future design of network input features.

Part of the work described in this thesis, namely dataset processing and network architecture design, was performed with Riemens (2015) and Kemper (2015).

2 Convolutional Neural Networks

Convolutional Neural Networks are biologically-inspired feedforward neural networks. CNNs feature multiple layers of various types that together make up a network's full architecture (LeCun, Bottou, Bengio, & Haffner, 1998). Thus, CNNs are a type of deep neural network. See Section 2.1 for an overview of layer types. In a CNN so-called *convolutional layers* (see Section 2.1.1) consist of one or more convolutional kernels. All convolutional kernels in a single layer contain an equal number of neurons in the same structural configuration. The kernels represent filters that respond to certain kinds of stimuli. Each kernel is then convolved over the input yielding a multidimensional output where values indicate the activation of the filter when applied at the corresponding location in the input.

This structure of convolutional layers with their convolutional kernels is related to the structure of mammalian visual cortices found by Hubel and Wiesel (1962). The visual cortex of mammals contains various cell arrangements. These cells respond to stimuli in specific regions of the visual field; their "receptive fields". These receptive fields are tiled over the entire visual field. Hubel and Wiesel found different cell types with distinct properties, such as some "simple cells" responding to straight lines in certain orientations in their receptive fields and some "complex cells" that were locally invariant to the location of the stimulus.

By stacking layers the input of layer $m > 1$ is the output of layer $m - 1$. The input of layer 1 is the input image. Through repeated convolution and down-sampling the filter features in the convolutional layers gradually become of higher-level and become more abstract (M. D. Zeiler & Fergus, 2014; Montavon, Braun, & Müller, 2010; Güçlü & van Gerven, 2014). For example, in a typical image classification task the low-level convolutional layers could respond to straight lines and other simple shapes. The next convolutional layer uses the output of the first layer and is able to respond to more complex shapes. This continues to the last convolutional layer which responds to, for example, human facial features.

CNNs have been applied to achieve a diverse set of goals, such as classification and feature extraction of images. Such networks have seen promising results especially with regard to image research. One of many such successful results was that of Krizhevsky et al. (2012), who trained a CNN on an ImageNet dataset consisting of 1.2 million images and 1,000 classes. They achieved a classification error rate of 17%, which was better than the previous state of the art.

2.1 Layer Types

A convolutional neural network consists of various layer types that form the full network. The layer types used in this research are described below.

2.1.1 Convolutional Layer

As the namesake of the type of neural network used here, the convolutional layer is what distinguishes this network from other types of deep neural networks. Each convolutional layer contains one or more equal-sized convolutional kernels $\mathbf{F}_n$. These kernels are filters of a certain dimension $h_{\mathbf{F}_n} \times w_{\mathbf{F}_n} \times d_{\mathbf{F}_n}$ that are convolved over the input; i.e., they are applied at each possible location (i, j) on the input. Here, the origin is the top-left element of the kernel. The full layer's filter bank, $\mathbf{F}$, is a stack of kernels $\mathbf{F}_n$. $\mathbf{F}$ has dimensions $h_{\mathbf{F}} \times w_{\mathbf{F}} \times d_{\mathbf{F}} \times N$, where N is the total number of kernels. When a kernel $\mathbf{F}_n$ is applied at location (y, x) of input $\mathbf{X}$, the result is calculated by function $f_n(y, x)$.

$$f_n(y,x) = \sum_{i=1}^{h_{\mathbf{F}_n}} \sum_{j=1}^{w_{\mathbf{F}_n}} \sum_{k=1}^{d_{\mathbf{F}_n}} \mathbf{X}_{i'j'k} \cdot \mathbf{F}_{nijk}$$

with $i' = i + y - 1$ and $j' = j + x - 1$.

Here, the depth of a kernel is equal to the number of feature maps the layer's input contains. Note that this does not have to be the case; in general, different kernel groups can be applied to varying dimensions of the input. The possible application positions for a kernel $\mathbf{F}_n$ on an input $\mathbf{X}$ range from $y = 1$ to $y = h_{\mathbf{X}} - h_{\mathbf{F}_n} + 1$ and from $x = 1$ to $x = w_{\mathbf{X}} - w_{\mathbf{F}_n} + 1$. An example of a convolutional kernel application can be seen in Figure 1.

The output of a convolutional layer consists of a number of feature maps. The number of output feature maps is equal to the number of kernels in that layer. Each feature map $\mathbf{Y}_n$ is the output of one of the kernels convolved over the input plus a bias, $\mathbf{b}_n$, that is added to all elements in the output. The full biases of a convolutional layer are given by $\mathbf{b}$. The feature maps' heights are $h_{\mathbf{Y}_n} = h_{\mathbf{X}} - h_{\mathbf{F}} + 1$. The widths are $w_{\mathbf{Y}_n} = w_{\mathbf{X}} - w_{\mathbf{F}} + 1$. Thus, the full output $\mathbf{Y}$ has dimensions $(h_{\mathbf{X}} - h_{\mathbf{F}} + 1) \times (w_{\mathbf{X}} - w_{\mathbf{F}} + 1) \times N$.

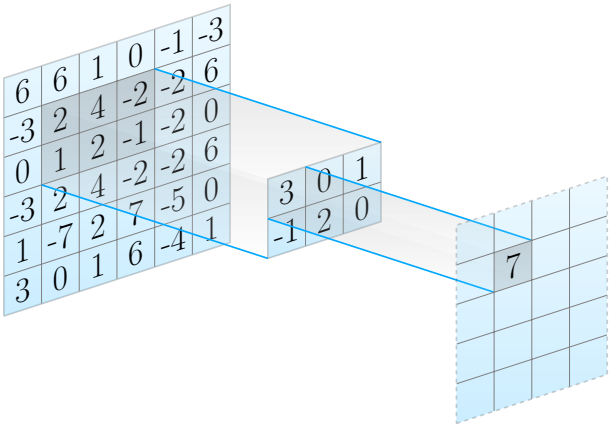


Figure 1: A convolutional kernel applied to location (2,2) of its input. The input consists of only one feature map. The value of the filter applied at this location is $(3 \cdot 2) + (0 \cdot 4) + (1 \cdot -2) + (-1 \cdot 1) + (2 \cdot 2) + (0 \cdot -1) = 7$.

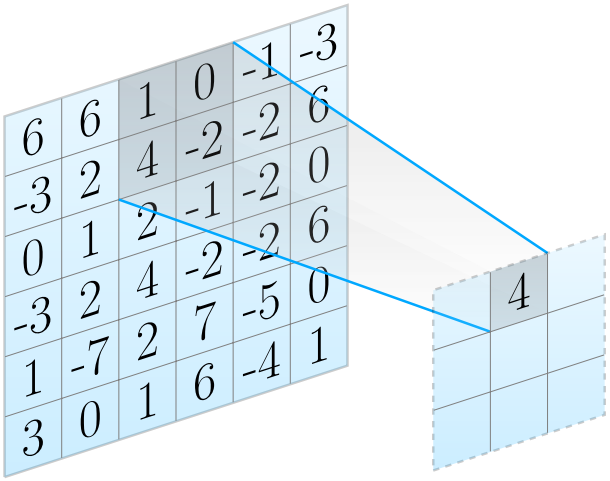


Figure 2: Max-pooling down-sampling applied on an input feature map. The value of max-pooling applied at this location is $\max(1, 0, 4, -2) = 4$.

2.1.2 Rectified Linear Units Layer

Rectified linear units (ReLUs) are simple neuronal units with an activation function $f(x) = \max(0, x)$, with x the input to the neuron. A ReLU serves as a linearity-breaking unit which allows deep network architectures to learn non-linear representations of the input. ReLUs have several advantages over other linearity-breaking activation functions, such as sigmoid functions, like sparse activations and biological plausibility (Glorot, Bordes, & Bengio, 2011).

In short, the output of a ReLU layer is the input of that layer with all negative values set to 0.

2.1.3 Max-pooling Layer

An important element of CNNs is down-sampling. Max-pooling layers perform non-linear down-sampling on the input feature maps. Each feature map is divided into (potentially overlapping) rectangles. For each rectangle, the maximal value is chosen as the output of that rectangle in the feature map.

See Figure 2 for an example of a max-pooling application.

2.1.4 Dropout Layer

CNNs, like most deep network architectures, suffer heavily from overfitting (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012). An overfitted classification network performs well on the training set, but poorly on a held-out test set. A method for improving generalization is to train multiple networks and combine the networks' predictions, such as by averaging (Wolpert, 1992). The obvious drawback of this approach is that it is computationally expensive to train multiple networks.

Another method for minimizing overfitting is to utilize *dropout*. With the dropout method hidden units are randomly selected and omitted. Effectively, the outputs of these hidden units – the values in the output feature maps – are clamped to 0. These units will not be updated in the following weight update step. During testing and other operation dropout is not performed. In these cases, the hidden units'

output values are downscaled in accordance with the dropout rate to account for the increased number of hidden units. Performing dropout prevents complex co-adaptions of the hidden units on the training data, because no hidden unit can rely on other hidden units being present (Hinton et al., 2012). Thus, hidden units will not only be useful in the context of other hidden units.

2.1.5 Fully Connected Layer

The role of classifier is fulfilled by fully connected layers (Simard, Steinkraus, & Platt, 2003). These layers consist of hidden units connected to every value in their inputs. At this point, all spatial information is lost and thus fully connected layers cannot be followed by convolutional layers. These layers can be seen as the "high-level reasoning" of a network. See Figure 3 for an example of two fully connected layers.

Each unit in a fully connected layer contains weights $\mathbf{F}_n$ and a bias $\mathbf{b}_n$. The full layer is a bank of hidden unit weights, $\mathbf{F}$, and biases, $\mathbf{b}$. The output $\mathbf{Y}_n$ of a hidden unit n is given as follows:

$$\mathbf{Y}_n = \mathbf{F}_n \circ \mathbf{X} + \mathbf{b}_n$$

Here, $\circ$ is the entrywise product operation. The full output of the layer, $\mathbf{Y}$, is the stack of hidden unit outputs $\mathbf{Y}_n$. Its dimension is $1 \times 1 \times N$ with N the number of hidden units. Fully connected layers can be seen as a special case of convolutional layers where the kernel dimensions are equal to the input dimensions.

2.1.6 Softmax Log-loss Layer

The softmax log-loss operator combines the softmax and log-loss operators. It is the objective function to be minimized. The operator is given by y.

$$y = -\sum_i \sum_j \left(\mathbf{X}_{ijc} - \log \sum_k e^{\mathbf{X}_{ijk}} \right)$$

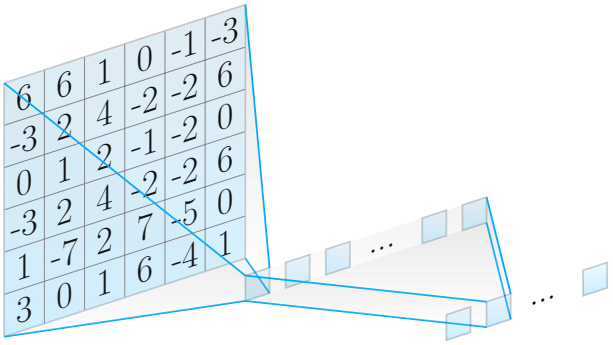


Figure 3: Example of two fully connected layers. The units in the first fully connected layer are connected to all units in the input layer. The units in the second fully connected layer are connected to all units in the first fully connected layer. In this figure the layers are implicitly followed by ReLU layers to break linearity.

Here, c is the ground truth class.

2.1.7 Padding and Stride

The convolutional layers (Section 2.1.1) and max-pooling layers (Section 2.1.3) can be configured with *padding* and *stride*.

Padding can be set for the top, right, bottom and left sides of the input $\mathbf{X}$ to a layer. Padding the right side of $\mathbf{X}$ by a value of three causes three new columns to be added to the right of $\mathbf{X}$. These columns are filled with zeros. This can be useful for both convolution and pooling, because without padding elements in the middle of the input will be seen more often by the convolution kernel or pooling window than elements at the border, given a stride of one. This is due to the convolution kernels and pooling windows not extending past the borders of the input.

Stride is the down-sampling factor of the convolutional and pooling layers. It can be configured separately for the horizontal and vertical dimensions. A stride of three for one of the dimensions causes the convolution kernel or pooling window to be applied at every third value in this dimension, starting from

the first. In effect the input will be down-sampled by factor tree in this dimension.

2.2 Learning

Learning a CNN classifier is performed by optimizing an objective function. The specific objective (loss) function used in this research is the softmax log-loss function (see Section 2.1.6). Because CNNs are usually applied to large-scale problems, optimization generally uses a variant of *stochastic gradient descent* (Bottou, 2010). In this research mini-batches are used for weight updates, which yield smoother weight updates relative to weight updates for single train samples.

In order to update the weights and biases of convolutional and fully connected layers the gradients have to be known for those layers. This is achieved by iteratively propagating the gradient known in layer m back to layer $m-1$, starting from the top layer, in a process called *backpropagation*. If the error E is the scalar output of the objective (loss) function, then the error gradient with respect to a certain layer's weights $\mathbf{F_l}$ can be computed by using the chain rule.

$$\frac{\partial E}{\partial \mathbf{f_l}} = \frac{\partial E}{\partial \mathbf{y_L}} \frac{\partial \mathbf{y_L}}{\partial \mathbf{y_{L-1}}} \cdots \frac{\partial \mathbf{y_{l+1}}}{\partial \mathbf{y_l}} \frac{\partial \mathbf{y_l}}{\partial \mathbf{f_l}}$$

Here, L is the number of layers in the network and $\mathbf{f_l}$ and $\mathbf{y_l}$ are the vectorized filters and outputs of layer l respectively. Practically, to calculate the gradient for a filter bank $\mathbf{F}$ the gradient of the error with respect to its output $\mathbf{Y}$ has to be known; $\partial E/\partial \mathbf{Y}$. This gradient is given by the layer above through its calculation of the gradient of the error with respect to its input; $\partial E/\partial \mathbf{X}'$. As follows from the network's architecture $\mathbf{X}' := \mathbf{Y}$ and thus $\partial E/\partial \mathbf{Y} = \partial E/\partial \mathbf{X}'$. For the softmax log-loss layer, which is the topmost layer, $\partial E/\partial \mathbf{Y} = 1$, as E is defined as the output of the loss layer. The gradient can then be propagated back through the layers by repeatedly calculating $\partial E/\partial \mathbf{X}$. To update filters $\mathbf{F}$ and biases $\mathbf{b}$ of a convolutional or fully connected layer, $\partial E/\partial \mathbf{F}$ and $\partial E/\partial \mathbf{b}$ can be calculated. For biases $\mathbf{b}$, vectorized filters $\mathbf{f} = \text{vec}(\mathbf{F})$ and vectorized output $\mathbf{y} = \text{vec}(\mathbf{Y})$:

$$\frac{\partial E}{\partial \mathbf{f}} = \frac{\partial E}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{f}}$$
$$\frac{\partial E}{\partial \mathbf{b}} = \frac{\partial E}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{b}}$$

For an overview of the derivatives of the specific layer types and the derivations thereof, see Riemens (2015).

Once the gradient of the error with respect to filter weights and biases are known, they can be updated in direction opposite to the gradient to improve network performance. The simplest update routine is performing a small step in direction of the negative gradient.

$$\mathbf{F}' = \mathbf{F} - \eta \frac{\partial E}{\partial \mathbf{F}}$$
$$\mathbf{b}' = \mathbf{b} - \eta \frac{\partial E}{\partial \mathbf{b}}$$

Here, η is the learning rate, and $\mathbf{F}'$ and $\mathbf{b}'$ are the new filters and biases.

3 Dataset

The TIMIT dataset of read speech was used to train a convolutional neural network (Garofolo et al., 1993). This dataset was designed for the development of speech recognition systems and other acoustic-phonetic studies. In this study, we are especially interested in TIMIT's labelled phone data.

3.1 Filtering and Mapping

The dataset contains three sentence types. Of these types, *SA*-type sentences are identical among all speakers, whereas the other two types (*SX* and *SI*)

are distinct between speakers. The *SX*- and *SI*-type sentences were designed to provide a good balance of phonemes, whereas *SA*-type sentences were designed to emphasize between-dialect differences. As per the recommendation of Lee and Hon (1989), the *SA*-type sentences were removed to prevent over-representing the phonemes contained in those sentences. Furthermore, fifteen allophones as identified by Lee and Hon (1989) were mapped to their corresponding phones, such as [m̩] to [m], effectively leaving 48 distinct phone class labels. In addition, similar phones in the resulting set of 48 phones, such as [ʌ] and [ə], were grouped to result in a set containing 39 phone labels. The network is trained to classify the 48 phones. The performance of the network is measured by using the grouping of 39 phones. This process has become a standard for training and evaluation on the TIMIT dataset (Lopes & Perdigão, 2011).

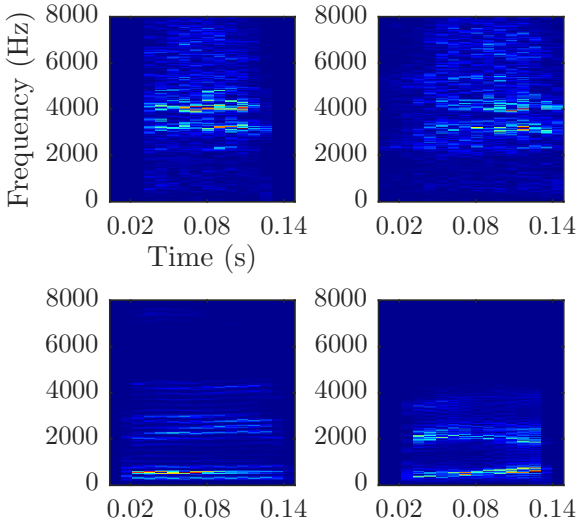


Figure 4: [ʃ] (top) vs [i] (bottom)

3.2 Preprocessing

The goal of this research is to gain insight into the internal representations of convolutional neural networks applied to the auditory domain. Specifically, I will attempt to reconstruct network inputs through network activation patterns. As this is exploratory research into a relatively new domain for convolutional neural networks, it is desirable to keep preprocessing simple.

Phones were extracted from the speech waveforms by slicing the sentence waveform audio recordings in correspondence with the provided phone time markers. This process results in audio waveform vectors for individual phones. From these individual phone waveforms, the vectors above the 95th percentile in length were discarded to minimize practical issues caused by inaccuracies in time marking of the data and other abnormalities. The remaining vectors were zero-padded, such that all waveforms were of a length equal to the length of the longest waveform. This resulted in waveforms of 2554 samples; or roughly 160 ms.

Afterwards, for every zero-padded phone waveform a complex-valued spectrogram was generated utilizing a short-time Fourier transform. The short-time Fourier transform was computed with a window size of 16.7 ms, an overlap of 7.6 ms and contained 400 sampling points. The resulting complex spectrograms were transformed into real spectrograms by omitting the imaginary (phase) values, leaving only amplitude values. These settings and transformations were chosen as they provided relatively good speech waveform reconstructions from the resulting spectrograms; they resulted in a maximal pairwise cross-correlation of roughly 0.7 between the original waveform and the waveform reconstructed from the spectrogram. Note that reconstructing with a larger number of sampling points strongly increases reconstruction accuracy. For example, reconstructing from 800 sampling points results in a maximal cross-correlation between the original waveform and the reconstruction of 0.99, but such large spectrograms slow down training due to hardware memory constraints. The generated spectrograms used in this research have dimension 201×16.

For input into the network, the spectrograms were normalized by subtracting the mean. Note that the spectrogram intensities were not normalized.

The resulting spectrograms for two different utter-

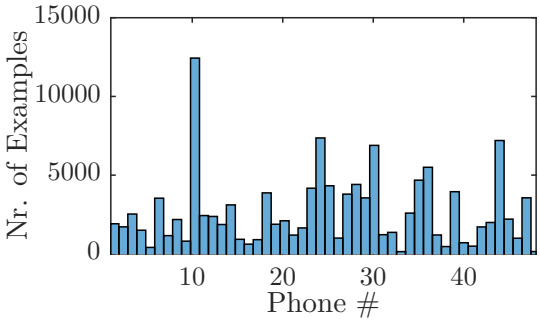


Figure 5: Histogram showing the phone distribution. Phone 10 (silence) accounts for 9.8% of all phones.

ances of two distinct phonemes can be seen in Figure 4. Clear similarities can be seen between the utterances of [ʃ], such as the cloud-like shapes. In contrast, for [i], both utterances have two distinct harmonics in the same frequency ranges. The utterances of the [ʃ] and [i] phonemes are clearly distinct in the time-frequency domain.

3.3 Resampling

The TIMIT dataset does not contain a uniform number of phone examples per class. The actual distribution of phones after folding and grouping can be seen in Figure 5. To create a uniform distribution for training and testing, phones were sampled from the full set each epoch. For training, 500 phones of each class were sampled from the training set. For testing, 200 phones of each class were sampled from the testing set. This sampling per phone was performed in two steps.

1. If the number of phones in a class set is less than the number to be sampled, all phones in the class set are repeatedly added to the sampled list until the number of phones left to be sampled is less than the number of phones present in the class set.

2. The number of phones left to be sampled are randomly picked without replacement from the phone class set.

This process was chosen as it ensured that phone class sets with few examples would always have all available examples present in each epoch. Contrast this with simple sampling without replacement where some phones in an already scarce class set would not necessarily be represented each epoch, effectively making the class even scarcer.

Next, the epochs were split into batches. The batches were generated by shuffling the sampled phones and splitting the sampled set into batches of 128 phones each.

4 Network Architectures

Much research has been done into CNNs applied to the visual domain. Much less research has been done into CNNs applied to the auditory domain. At the time of writing no consensus has been reached as to which network architectures work best for phone classification. Different authors recommend different architectures, but in general architectures in the auditory domain can be classified as either performing *frequency convolution* or *time convolution*.

There is an important difference between CNNs applied to the visual domain and CNNs applied to spectrograms in the auditory domain. In the visual domain, convolutions across the horizontal and vertical axes carry similar meaning. However, in the auditory domain these convolutions are not comparable (Tóth, 2014).

Applying convolution across the frequency axis makes the network less sensitive to variance in the different formant positions of the same phone between different speakers. To this end, authors such as Abdel-hamid et al. (2012) and Sainath, Mohamed, Kingsbury, and Ramabhadran (2013) have applied frequency convolution to capture translational invariance. With this property frequency convolution CNNs consistently outperformed deep neural networks (DNNs) on the same task (Tóth, 2014).

In contrast, applying convolution across the time axis carries less obvious advantages. Nonetheless,

Convolution type	Depth	Accuracy
Time	2, 2	48.7%
Frequency (large kernels)	2, 2	56.4%
Frequency	2, 2	59.1%
Frequency (full padding)	4, 3	61.8%
Frequency (overlapping pooling windows)	3, 4	62.6%

Table 1: Overview of the best-performing network architectures that were tested. The depth indicates the number of convolutional layers followed by the number of fully connected layers.

Veselý, Karafiát, and Grézl (2011) have applied a neural network architecture based on CNNs that successfully utilizes time axis convolution. On the other hand, Abdel-hamid, Deng, and Yu (2013) evaluated various CNN architectures and training techniques, including time axis convolution and both time and frequency axis (2D) convolution. They found that frequency convolution performed best, followed by 2D convolution. In their experiments, time convolution performed worst. In addition, Sainath, Kingsbury, et al. (2013) tested architectures utilizing time axis pooling without subsampling. They achieved only a slight gain, and only by utilizing stochastic pooling in time (see M. Zeiler and Fergus (2013)).

Though not the goal of our research, we have performed some preliminary experiments with differing CNN architectures. Our results can be seen in Table 1. The best-performing architecture was designed to be as simple as possible with few hidden units per layer. The network was tested both with and without first and second order derivative channels added to the input spectrogram, but these derivative channels made no difference in classification performance. Due to its performance and simplicity, we decided to use this network, without derivative channels, for the experiments.

4.1 Used Architecture

The network we used performed convolution along the frequency axis. Its architecture consisted of three convolutional layers followed by four fully connected layers. Each convolutional layer was followed by a ReLU layer and a max-pooling layer. The max-pooling layers all had a pooling size of 5 over the frequency axis and a stride of 2. All fully connected layers were followed by ReLU layers, except for the last; the last fully connected layer was followed by the softmax log-loss operator. For an overview of the network, including feature map sizes, see Figure 6.

5 Implementation

We used *MatConvNet* for training CNNs (Vedaldi & Lenc, 2014). MatConvNet is a CNN framework for MATLAB, based on *Caffe*. MatConvNet has built-in CUDA-support to exploit the parallelization optimizations of GPUs. In addition to the basic CNN functionality provided by MatConvNet, we have implemented several extensions on top of the framework.

5.1 Asynchronous Pre-fetching

In a purely single-threaded training algorithm time is lost by waiting for the next batch of images to be loaded from the hard drive. The training cannot continue until the batch is ready. As such, it is desirable to have the next batch loaded in the background during the processing of the current batch.

MatConvNet has support for such asynchronous pre-fetching, but its implementation requires the use of JPEG images. This is not ideal for spectrogram inputs as the JPEG file format generally introduces compression artefacts. To overcome this, we wrote a custom training wrapper that calls a MATLAB C++-script able to load spectrograms from simple, uncompressed binary files.

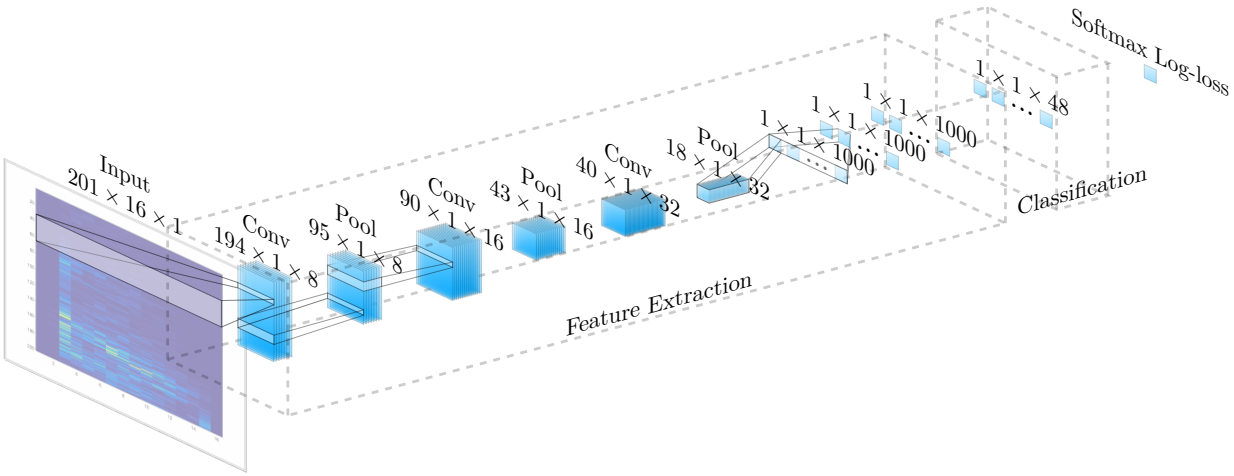


Figure 6: The used network's architecture. In this figure, ReLUs are implicitly placed after the convolutional and fully connected layers, with exception of the last fully connected layer. The dimensions indicate feature map sizes. Source and destination localities of some of the calculations performed by the different layers are shown.

5.2 Adaptive Gradient

The Adaptive Gradient algorithm (AdaGrad) is a theoretically sound method for automatically adapting the learning rate (Duchi, Hazan, & Singer, 2011). Besides it replacing problem-dependent learning rate schedules with a much less sensitive master step-size, it adaptively updates the learning rate per feature. Learning rates for frequent features quickly decrease, whereas learning rates for rare features remain high. This means that even infrequent features can be effectively updated.

AdaGrad works by keeping track of the sum of squared historical gradients. Then, the filter and bias update steps are changed.

$$\mathbf{F}' = \mathbf{F} - \boldsymbol{\lambda}^{\mathbf{F}} \circ \left(\frac{\partial E}{\partial \mathbf{F}} \right)$$
$$\mathbf{b}' = \mathbf{b} - \boldsymbol{\lambda}^{\mathbf{b}} \circ \left(\frac{\partial E}{\partial \mathbf{b}} \right)$$

with:

$$\boldsymbol{\lambda}^{\mathbf{F}}{}_{ijkn} = \frac{\eta}{\sqrt{\mathbf{G}^{\mathbf{F}}{}_{ijkn}}}$$
$$\boldsymbol{\lambda}^{\mathbf{b}}{}_{n} = \frac{\eta}{\sqrt{\mathbf{G}^{\mathbf{b}}{}_{n}}}$$

Here, $\mathbf{F}'$ and $\mathbf{b}'$ are the new filters and biases, $\mathbf{G}^{\mathbf{F}}$ is the sum of squared historical gradients with respect to the filter bank, $\mathbf{G}^{\mathbf{b}}$ is the sum of squared historical gradients with respect to the biases, and η is the master step-size.

5.3 Regularization

L_2 regularization is a method of preventing overfitting. Here, it penalizes filters with high weight values. For an example of the mathematical details of L_2 regularization, see Section 6.4.1. The implementation details of the regularization applied to filter

learning are almost the same as described in that section. However, instead of applying regularization on the input the regularization is applied to the individual filter banks. In general, L_2 regularization drives models towards sparse weights (Ng, 2004).

We have tested L_2 regularization as a means of decreasing overfitting. Though it did decrease overfitting, in our case it only did so by increasing the error on the train set. It did not improve the classification accuracy achieved on the test set.

6 Input Reconstruction

To gain insight into how CNNs are able to perform well in the auditory domain, it is interesting to investigate the representation of the domain learned by the network. One way of doing this is by looking at the actual learned features in the various layers – it would be expected to see certain spectrogram features such as various formants represented in higher-level convolutional layers. See the research performed by Kemper (2015) for more insight into this topic. Another way to investigate the internal representation is by attempting to reconstruct an input into the network through looking at the patterns of activation caused by that input. Reconstruction does not generate an actual reconstruction of the input; it generates a reconstruction of how the network perceived the input. As such, insight can be gained into how the network represents the input. It is this approach that will be expanded upon in this thesis.

6.1 Objective

Given an input $\mathbf{X}$ into a network, a series of hidden unit activations $\mathbf{Y_1}, \mathbf{Y_2}, ..., \mathbf{Y_L}$ will result. Here, $\mathbf{Y_n}$ is the output of layer n and L is the number of layers in the network. It would be expected that for a network with a good representation of the domain the activations of the convolutional layers can be used to generate a reasonably good reconstruction. However, from the fully connected layers onward spatial information is lost. As such, from those layers it is unlikely that a good reconstruction can be made.

Theoretically, any subset of information contained in the entire output set of $\mathbf{Y_1}, \mathbf{Y_2}, ..., \mathbf{Y_L}$ (including partial layer-wise output information) could be used to generate a reconstruction. Nonetheless, this research is constrained to looking only at the full output $\mathbf{Y}_n$ of a single layer per reconstruction.

6.2 Methods

Various approaches to input reconstruction can be taken, such as simulated annealing, error backpropagation to the input, and network decoding. The goal in all approaches is to minimize the reconstruction error. In this research the error is quantified as the euclidean distance R between the original input $\mathbf{X}^*$ and the reconstructed input $\mathbf{X}$.

$$R = \sqrt{\sum_i \sum_j \sum_k \left(\mathbf{X^*}_{ijk} - \mathbf{X}_{ijk}\right)^2}$$

6.3 Simulated Annealing

Preliminary experiments were performed for input reconstruction by simulated annealing. In general, simulated annealing is a probabilistic global optimization algorithm. It is able to explore the entire solution space of the problem and probabilistically moves uphill and downhill in the error space (Corana, Marchesi, Martini, & Ridella, 1987; Goffe, Ferrier, & Rogers, 1994). Each step, a new reconstruction $\mathbf{X}'$ is generated from the current reconstruction $\mathbf{X}$ by adding random noise (see Section 6.3.1). The objective function value E' of $\mathbf{X}'$ (see Section 6.3.2), is then compared with the objective function value E of $\mathbf{X}$. If $E' < E$, then $\mathbf{X}'$ will be set as the new reconstruction. If $E' \geq E$, $\mathbf{X}'$ will be set as the new reconstruction with probability p.

$$p = \exp\left(\frac{E - E'}{T(k)}\right)$$

$T(k)$ is the temperature at update step k. It is initially set to a high value and is then gradually decreased according to an annealing schedule. Generally, $T(k_{\max}) \approx 0$.

6.3.1 Neighbour Generation

The way in which neighbours are generated is of importance for the efficacy of the algorithm. Generally, phone spectrograms have high activation in a restricted frequency band. As such, I propose a method for generating neighbours that makes use of this property. It is true that not all phones share this property, but it makes for a good approximation for the majority of phones.

To generate a neighbour, one first generates a matrix $\mathbf{Q}$ of dimension $n \times w$. Here, n is the number of consecutive frequency bands the neighbour should differ in and w is the spectrogram width. This matrix should have a certain percentage of its values v set uniformly in the range $-r \leq v \leq 0$ or in the range $0 \leq v \leq r$. The other values are set to 0. I used 50% with $r = 0.05$. Then, $\mathbf{Q}$ is added entrywise to a band of rows in $\mathbf{X}$ of equal size. The resulting matrix is the neighbour $\mathbf{X}'$. To make sure neighbours with changes near the top and bottom borders are also frequently generated, $\mathbf{Q}$ can be added expanding past the borders of $\mathbf{X}$. The overflowing rows are ignored.

If simulated annealing is performed in a network trained to classify spectrograms with delta channels, special care has to be taken. To limit the search space, it is advisable to generate neighbours only for the static (non-delta) spectrogram and calculate the new delta channels from that.

6.3.2 Objective Function

The objective function E used for reconstruction through simulated annealing is equivalent to that of backpropagation to the input (see Section 6.4). Thus, the function to be minimized is given by E.

$$E = \sum_i \sum_j \sum_k \left(\mathbf{Y}^*_{\mathbf{n}ijk} - \mathbf{Y}_{\mathbf{n}ijk} \right)^2 + \lambda ||\mathbf{X}||_2^2$$

6.4 Backpropagation to the Input

By performing a forward pass through the network with input $\mathbf{X}^*$, the output activations $\mathbf{Y}^*_{\mathbf{1}}, \mathbf{Y}^*_{\mathbf{2}}, ..., \mathbf{Y}^*_{\mathbf{L}}$ are generated. If reconstruction is to be performed with respect to the activations of layer n, the goal is to create a reconstruction by continuously updating the current-best reconstruction with respect to this layer's activations. The updating is done in such a way as to minimize the difference between the activations caused by the reconstruction and the activations caused by the original input. In this way, the activations will start to look alike which, hypothetically, means the reconstruction will start to look like the original input. Formally, the objective function that is to be minimized for reconstructing the input is E.

$$E = \frac{1}{2} \left(\sum_i \sum_j \sum_k \left(\mathbf{Y}^*_{\mathbf{n}ijk} - \mathbf{Y}_{\mathbf{n}ijk} \right)^2 \right)$$

To perform the actual reconstruction, the network is continuously fed the current-best reconstruction. At the end of the feedforward pass (to the required layer) the derivative of the objective function is calculated with respect to that layer.

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{Y}_{\mathbf{n}ijk}} &= \frac{\partial \frac{1}{2} \left(\mathbf{Y}^*_{\mathbf{n}ijk} - \mathbf{Y}_{\mathbf{n}ijk} \right)^2}{\partial \mathbf{Y}_{\mathbf{n}ijk}} \\ &= \mathbf{Y}^*_{\mathbf{n}ijk} - \mathbf{Y}_{\mathbf{n}ijk} \end{aligned}$$

In matrix form:

$$\nabla E = \frac{\partial E}{\partial \mathbf{Y}_{\mathbf{n}}} = \mathbf{Y}^*_{\mathbf{n}} - \mathbf{Y}_{\mathbf{n}}$$

This gradient is propagated back to the input through normal backpropagation.

6.4.1 Regularization

Updating the reconstruction is performed through regular gradient descent updates enhanced by a bold driver factor (see Section 6.4.2). Spectrograms are generally sparsely activated; only having activations in a small frequency range. Because of this, L_2 regularization is eminently useful to ensure reconstructions are sparsely activated as well. To perform L_2 regularization, an error term θ is added to the objective function E (Ng, 2004).

$$\theta = \frac{\lambda}{2}||\mathbf{X}||_2^2 = \frac{\lambda}{2}\left(\left(\sum_i \sum_j \sum_k (\mathbf{X}_{ijk})^2\right)^{1/2}\right)^2$$

Thus,

$$\theta = \frac{\lambda}{2}\sum_i \sum_j \sum_k (\mathbf{X}_{ijk})^2$$

To minimize this part of the error function, the gradient of θ with respect to the input $\mathbf{X}$ has to be known.

$$\begin{aligned}\frac{\partial \theta}{\partial \mathbf{X}_{ijk}} &= \frac{\lambda}{2}\frac{\partial (\mathbf{X}_{ijk})^2}{\partial \mathbf{X}_{ijk}} \\ &= \lambda \mathbf{X}_{ijk}\end{aligned}$$

In matrix form:

$$\nabla\theta = \frac{\partial \theta}{\partial \mathbf{X}} = \lambda \mathbf{X}$$

Then, the full gradient with respect to the input becomes:

$$\nabla E = \frac{\partial E}{\partial \mathbf{x}} = \frac{\partial E}{\partial \mathbf{y_1}}\frac{\partial \mathbf{y_1}}{\partial \mathbf{x}} + \lambda \mathbf{x}$$

with $\mathbf{x}$ and $\mathbf{y_1}$ the vectorized forms of $\mathbf{X}$ and $\mathbf{Y_1}$ respectively.

6.4.2 Bold Driver

Reconstruction was tested utilizing both simple stochastic gradient descent and gradient descent with AdaGrad. The simple stochastic method yielded inconsistent training results when used for reconstructing from different layers. This is likely due to different orders of magnitude of the backpropagated gradients. Optimizing the meta-parameter η per layer would be time-consuming and would not be robust to changes. AdaGrad was tested as a simple replacement for stochastic gradient descent. Though it performed better, the master stepsize still had to be low in order for the reconstruction to converge when utilizing higher layer activations. This made learning unnecessarily slow for earlier layers. Furthermore, it is not clear when a reconstruction has converged enough such that updating can be stopped; convergence from higher layers took on the order of 10 to 100 times more batch updates than from lower layers.

Instead of attempting to optimize the various parameters, a different learning rate adaptation technique was used, termed the *bold driver* (Battiti, 1989). The bold driver is a factor λ with which the master learning rate η is multiplied to obtain the current-batch learning rate. The λ factor is continuously updated throughout the updating progress by multiplying it with a factor $p_{\Uparrow} > 1$ if the error value is lower than the previous batch. If the error value is higher, the factor is multiplied by $p_{\Downarrow} < 1$. Suitable values are, for example, $p_{\Uparrow} = 1.01$ and $p_{\Downarrow} = 0.9$.

In effect, the learning rate is constantly increased slightly, until the gradient is overshot and oscillations would start to occur; then, the learning rate is slashed aggressively. Once a reconstruction has converged, further updating simply oscillates around the convergence point. As such, the error value would be oscillating as well. Due to this, a converged reconstruction is indicated by a low bold driver factor λ. Thus, we can look at the bold driver factor to decide when to stop updating. For this problem, a suitable cut-off point was $\lambda < 0.005$ (with a master learning rate $\eta = 10^{-6}$).

6.5 Decoder Network

Autoencoders are a type of neural network that are trained to encode a certain input in such a way as to be able to decode that representation. Thus, the output target of such a network is the input itself (Bengio, 2009). The application of autoencoders is closely related to the problem discussed here. However, instead of training the encoder and decoder simultaneously, the encoder is given; it has been trained as part of a classification network. Thus, the goal is to create a decoder for our encoder that is able to reconstruct the input into the encoder from the representation generated by the encoder.

To this end, a new encoder network was trained that is highly similar to the original network used throughout this research. The new network featured padding for the convolutional layers such that no border information was lost. This is necessary to be able to compute the convolutional transpose for decoding whilst preserving the same input dimensionality.

6.5.1 Convolution Transpose

The convolutional transpose operator is, in the case of full-padding without stride, simply regular convolution. For autoencoders, often "tied weights" are used. Then, the filter bank of a convolutional decoder layer is equal to the transposed filter bank of the corresponding convolutional encoder layer. Such a construct is not suitable for the case of training a decoder given an encoder, because tied weights require the two filter banks to be trained to a symmetric solution (Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010), but in this case the encoder will not be trained further.

6.5.2 Unpooling

The encoder performs pooling to sub-sample the input. The decoder has to reverse this by up-sampling through unpooling. There are a variety of methods with which unpooling can be performed. One method

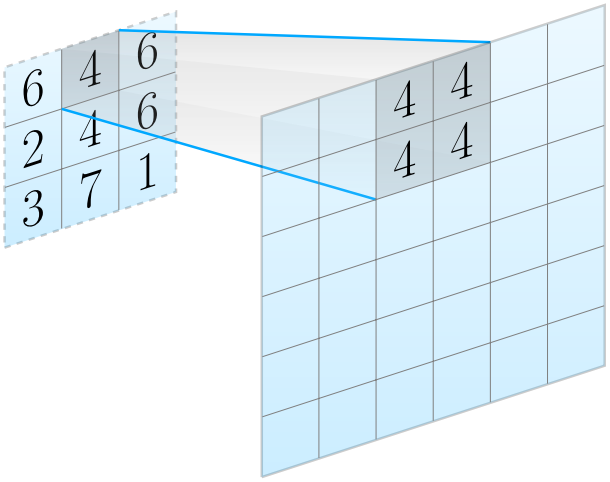


Figure 7: Blind-unpooling up-sampling applied on an input feature map.

is to remember the locations of the maximal values chosen by max-pooling. Then, during unpooling, *switch-unpooling* is used such that the values are set at the remembered locations. The empty locations are set to 0. However, it turned out that this method undesirably leaks information from early encoding layers to late decoding layers (see Figure A.1a). In fact, setting all the values of the filters and biases of the first decoding convolutional layer to 0, creating a mangled network, still produces a good reconstruction when using switch unpooling. This indicates that a decoder using this method has more information available than only the encoded representation of the input. As such, switch-unpooling is not suitable for our particular case of training a decoder given an encoder.

Blind-unpooling is another method by which unpooling can be performed. An example of this method can be seen in Figure 7. This method is entirely uninformed. It takes the values of the input feature map and repeats those values onto the output at the relevant locations (as determined by the pooling configuration). If the pooling windows overlapped, blind-pooling sets the value at an output location to the average of the values that are to be

set at that location. In contrast to switch-unpooling, this method does not leak any information (see Figure A.1b).

6.5.3 Objective Function

The objective function E for an autoencoder/decoder is given by half the squared euclidean distance between the output $\mathbf{X}'$ and the input $\mathbf{X}^*$.

$$E = \frac{1}{2}\left(\sum_i \sum_j \sum_k \left(\mathbf{X}^*{}_{ijk} - \mathbf{X}'{}_{ijk}\right)^2\right)$$

Thus, the gradient of the error with respect to the last layer's output $\mathbf{Y} := \mathbf{X}'$ is given by:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{Y}_{ijk}} &= \frac{1}{2}\frac{\partial \left(\mathbf{X}^*{}_{ijk} - \mathbf{Y}_{ijk}\right)^2}{\partial \mathbf{Y}_{ijk}} \\ &= \mathbf{X}^*{}_{ijk} - \mathbf{Y}_{ijk}\end{aligned}$$

In matrix form:

$$\nabla E = \frac{\partial E}{\partial \mathbf{Y}} = \mathbf{X}^* - \mathbf{Y}$$

This gradient is the topmost gradient given to the network. The gradient is then propagated backwards as usual.

6.5.4 Updating

The goal is to train a decoder for the encoder, and not an autoencoder initialized with a classification encoder. Thus, during the weight and bias updates, only the weights and biases of the decoder part of the network are updated.

6.5.5 Stacked Decoder

An issue encountered with training the decoder was that the gradient for a randomly initialized decoder seemed to push the decoder into the direction of mapping all inputs to silence. This is likely caused by the fact that phone spectrograms have sparse activations, and thus silence is a relatively good reconstruction. To overcome this, I implemented a decoder training procedure based on that of stacked autoencoders.

A stacked autoencoder is trained in a greedy layer-wise fashion (Bengio & Lamblin, 2007). First, a (two-layer) autoencoder is trained on the raw inputs $\mathbf{X}$ to learn encoder features $\mathbf{F}$. The encoder output is then mapped to the network output by the decoder. Once these layers are trained, a second autoencoder is added between the encoder and the decoder of the first autoencoder. Thus, the inputs to the new autoencoder is the output of the first autoencoder's encoder, and its output is the input to the first autoencoder's decoder. This second autoencoder is then trained in a similar fashion.

For the case of training only a decoder, this process cannot be used directly, as the encoder is already given. However, a similar process can be used. Instead of training the full decoder at once, the first decoder layers (i.e., the layers near the middle of the network) can be trained first. Here, the target output of a decoder layer is the input to the corresponding encoder layer. The same objective function as in Section 6.5.3 can be used, except that the input $\mathbf{X}^*$ and output $\mathbf{Y}$ are now the input to the corresponding encoder layer and output of the decoder layer respectively. This process is carried out iteratively to the last layer. Once all decoders have been trained one by one, all decoders are trained simultaneously for fine-tuning.

6.6 Decoder Network & Backpropagation

One use of autoencoders is to generate a pretrained prior for training a classifier. To do this, the autoen-

coder is trained whereafter the decoder is omitted and a classifier is put in its place. Then, the classifier is trained and the encoder is fine-tuned with regular backpropagation. This often produces good results, as it helps prevent the encoder's gradient descent from getting stuck in a globally poor local minimum (Erhan, Bengio, & Courville, 2010).

Taking from this idea, it is conceivable that using the trained decoder's output as described in section 6.5 and fine-tuning it with backpropagation as described in Section 6.4 would similarly result in improved reconstruction performance. Formally, take $\mathbf{X} = D(\mathbf{X}^*)$ as the decoded output of the network's representation of $\mathbf{X}^*$. Take $\mathbf{X} = B(\mathbf{X}^*, \mathbf{X}')$ as the output of the backpropagation algorithm when reconstructing $\mathbf{X}^*$ and starting from the reconstruction $\mathbf{X}'$. Then, $\mathbf{X} = B(\mathbf{X}^*, D(\mathbf{X}^*))$ is the reconstruction generated from the chained operation.

6.7 Averaging

Each single reconstruction that is not arbitrary will be some combination of a good reconstruction of the input and noise. In these cases, it can be advantageous to generate multiple reconstructions and take the average to arrive at the final reconstruction. It would be expected that the reconstruction error of the resulting average is lower than the average reconstruction error among single reconstructions. This is mostly due to noise being averaged away.

For deterministic algorithms, generating different reconstructions for the same input requires special care. The steps taken for the individual algorithms are described below.

Simulated Annealing Because this algorithm is probabilistic by nature, it inherently supports averaging.

Backpropagation Outputs of backpropagation can easily be used for averaging, as long as the reconstruction initialisation is non-deterministic. This is generally the case, as gradient descent requires a symmetry-breaking initialisation.

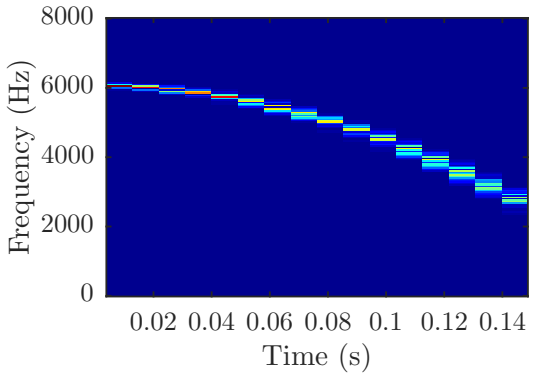


Figure 8: A non-speech spectrogram

Decoding Decoding can be made non-deterministic by adding slight noise to the input, causing slightly differing outputs. However, for a good encoder/decoder-pair it would be expected that input noise has little effect on the reconstruction.

7 Experiments

The four described reconstruction methods were tested on network layers 1, 3, 6, 9 and 12 (L1, L3, L6, L9 and L12). These layers correspond to the first convolutional layer, the first convolution-relu-pool triplet, the second such triplet, the third such triplet and the second fully-connected layer respectively. For the decoding method five decoders were trained to reconstruct representations generated by the encoder up to these layers.

To test whether the network has actually learned a speech representation – which implies that not just any input can be reconstructed – non-speech spectrograms were generated. These spectrograms were tested for reconstruction performance as well as phone spectrograms from the test set. These non-speech spectrograms were generated from chirp waveforms. An example of such a chirp spectrogram can

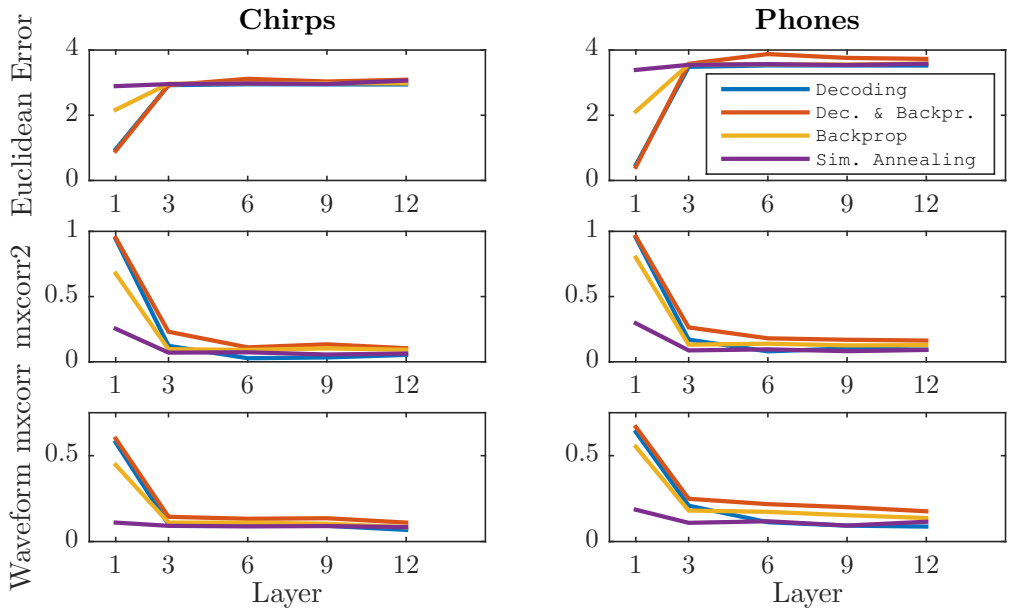


Figure 9: The average reconstruction errors and correlations, and waveform correlations per layer. The top figures show the euclidean reconstruction errors, the middle figures show the maximal spectrogram cross-correlations, and the bottom figures show the maximal waveform cross-correlations.

be seen in Figure 8.

Aside from qualitatively measuring reconstructions by visually judging similarity between the reconstruction and the original spectrogram, at least three quantitative reconstruction performance measures are available:

1. Euclidean distance between the reconstruction and the original spectrogram;

2. Maximal matrix cross-correlation (mxcorr2) between the original spectrogram and the reconstructed spectrogram, calculated as the maximal xcorr between the vectorized matrices; and

3. Maximal vector cross-correlation (mxcorr) between the original waveform and the reconstructed waveform.

Due to computational limitations, the reconstruction experiments were ran on only one randomly chosen example from each of the 48 phone classes.

7.1 Results

7.1.1 Quantitative

As can be seen in Figure 9, reconstructing from the first layer results in a quantitatively better reconstruction than later layers. Furthermore, the waveform mxcorr appears to decrease as the reconstructions are generated from deeper layers. Interestingly, with regard to the waveform mxcorr, reconstructions from the second fully connected layer (layer 12) are only slightly worse than reconstructions from the late convolutional layers. Furthermore, with regard to the waveform mxcorr seen in the Figure 9, reconstructions for non-chirp spectrograms were of better quality than those of chirp spectrograms.

The average waveform mxcorr of the average waveform with phone waveforms is 0.12. I name this the "waveform mxcorr threshold". The average waveform is constructed from the average spectrogram. Once the waveform mxcorr of a reconstruction drops

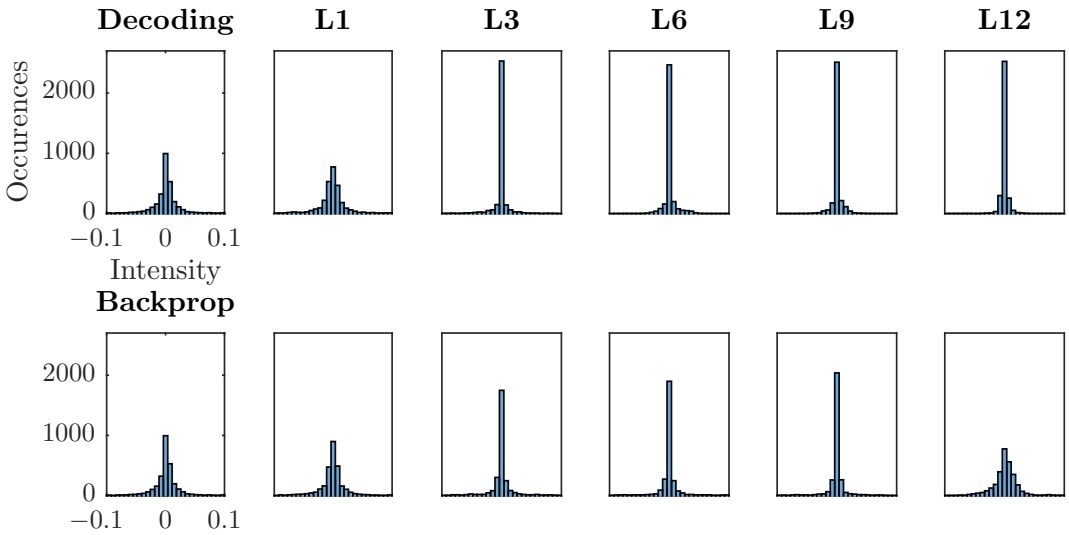


Figure 10: Spectrogram intensity histograms for the original phone spectrogram and the reconstructed spectrograms for one case of phone [ɔ]. In the left-most column the intensity histograms of the original inputs into the network are shown (these are identical over the methods). The other columns show the intensity histograms of the reconstructions generated from a certain layer by the corresponding method. Only two methods are shown.

below this threshold, it is indicated that the average waveform is, on average, a better reconstruction with regard to the waveform mxcorr.

Almost all reconstruction methods for chirps are below the threshold from layer 3 onward. The only exception is the method of decoding followed by backpropagation. This method drops below the threshold at layer 12. For phones, backpropagation and decoding followed by backpropagation are above the waveform mxcorr threshold for all layers. Simple decoding and simulated annealing drop below the threshold at layers 6 and 3 respectively.

From L3 onward, the reconstructed spectrogram intensities are greatly reduced relative to their original inputs (see Figure 10).

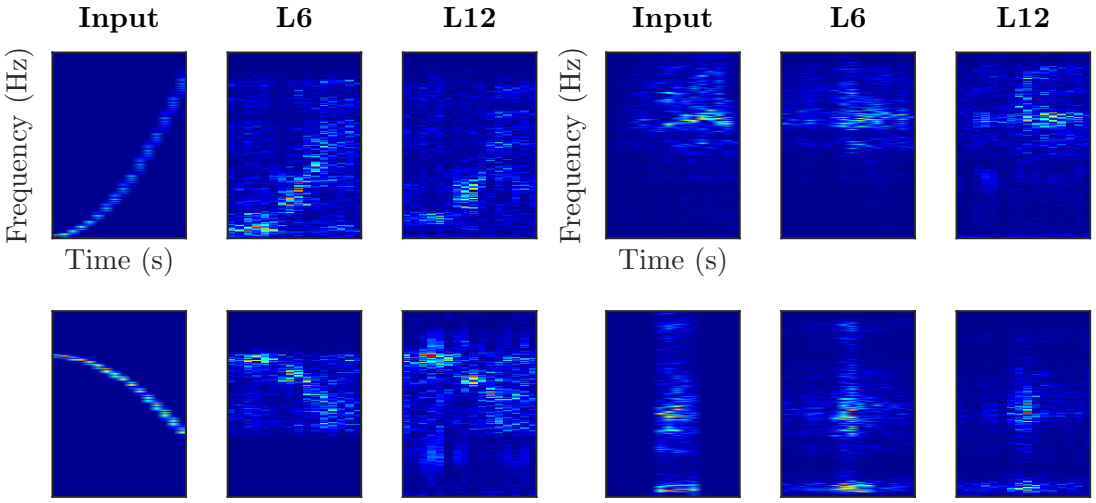
7.1.2 Qualitative

Visually, the method of decoding followed by backpropagation seems to reconstruct the phone spectrogram best. Second-best is backpropagation, especially from L6 onward. Simulated annealing performs worst over all layers. This is in agreement with the quantitative results found when looking at the waveform mxcorr. All reconstructions for a single phone generated by the used methods for all layers can be seen in Figure B.1.

Figure 11 shows the reconstruction of two chirps and two phones generated by the method of decoding followed by backpropagation. It appears that phones are reconstructed slightly better than chirps. Additionally, the activations in the reconstructions appear to be slightly smeared across the time domain. This is more apparent in Figure B.2.

8 Discussion

I have shown that reconstructions are nearly perfect for the first layer for both the decoding network and the decoding network followed by backpropagation.



(a) Reconstructions generated for chirp spectrograms. (b) Reconstructions generated for phone spectrograms.

Figure 11: Comparison of reconstructions generated for chirps and reconstructions generated for phones. The reconstruction method that was used to generate these reconstructions is decoding followed by backpropagation. In both figures each row represents the original input, the reconstructions generated from the sixth layer, and the reconstructions generated from the twelfth layer respectively.

This is not surprising; at this point merely a single convolution has been performed on the input. No sub-sampling has been performed yet. As such, the decoder merely has to learn a good linear inverse of the convolutional layer. Backpropagation (without first decoding) and simulated annealing do not reach this performance because they do not exploit the inversibility of the linear transformation.

Phone reconstructions were of higher quality than chirp reconstructions. This reinforces the belief that the encoder (i.e., excluding the fully-connected layers; the classifier) learns to generate a more compact representation of speech audio.

I have demonstrated that simulated annealing performs worse than all other methods all-round. This is likely caused by a combination of factors. Firstly, the specific neighbour generation algorithm makes assumptions about the general form of phone spectrograms (see Section 6.3.1). This assumption does not necessarily hold for all phone spectrograms. Indeed, some phones only have activations in a limited part of the time domain. Secondly, the algorithm was performed with a restricted number of update steps. Increasing this number likely increases performance, but reconstruction would become computationally prohibitive; for each step the network activations have to be evaluated. Thirdly, simulated annealing does not make use of the structure of the network like the decoder and backpropagation methods do. As the search space is essentially infinite, simulated annealing would likely only ever be able to give a rough estimate of the reconstruction in this domain.

The reconstruction methods generally generated reconstructions with intensity smeared across the time domain. Because reconstruction from activity patterns does not actually reconstruct the input, but rather the input as perceived by the network, this tells us that the network's features probably span the entire time domain. This seems to be in agreement with the results found by (Kemper, 2015). Moreover, this behaviour of the network is sensible. Phones do

not change meaning whether they are spoken quickly or slowly. Thus, the phone class should not change depending on the phone's width in the time domain.

From layer 3 onward, the reconstructions had vastly reduced intensities relative to the original inputs. It is conceivable that this is caused by L2 regularization. However, this intensity discrepancy holds true as well for outputs generated by the decoder networks; as such, it is likely that this is a result of the network's internal representation of the input. A phone spoken softly or loudly makes no difference to the actual class of the phone. With this in mind, it seems logical that the network does not make use of intensity information for its internal representations.

The reconstructions from the second fully connected layer were of almost equal quality as reconstructions from the late convolutional layers. This indicates that not all spatial information is necessarily lost in the layers where the network performs its high-level reasoning.

8.1 Quantitative Measures

Using the three quantitative measures described in the experiments section (Section 7) to objectively judge reconstruction quality is troublesome. Firstly, all measures are unnuanced. This holds true especially for both the euclidean spectrogram error and the maximal spectrogram cross-correlation.

The discrepancy between reconstruction intensity and input intensity partly explains the unpredictiveness of the euclidean error with regard to reconstruction quality. The intensity of reconstructions approaches the intensity of the zero-spectrogram. As such, the euclidean error approaches the euclidean distance of spectrograms to the zero-spectrogram. Thus, in this case, euclidean errors as-is are meaningless measures for reconstruction performance from layer 3 onward.

Reconstructions for higher layers generated by simulated annealing – the worst method – had an average waveform mxcorr close to that of decoding followed by backpropagation – the best method. Qualitatively, decoding followed by backpropagation performed profoundly better than simulated annealing. It seems that the waveform mxcorr should be interpreted more as an ordinal measure than as a cardinal one; a reconstruction with a waveform mxcorr of 0.2 is not "two times better" than a reconstruction with a waveform mxcorr of 0.1. With this in mind, it appears nonetheless that the waveform mxcorr provides the best quantitative measure of reconstruction quality; ordinally it agrees with qualitative assessment, and it the only quantitative measure that provides clear nuance.

8.2 Limitations

It has been attempted to make this research as rigorous as possible. Many experiments, architectures and methods have been considered in detail. Nevertheless, this research comes with some limitations.

1. It is possible that simulated annealing can be improved by better neighbour generation. However, this requires the algorithm to be highly tailored to some model of phone spectrograms. In some sense this already is the issue that is attempted to be solved through encoding and reconstruction in the first place.

2. CNNs are deep architectures. They are best used with a very large dataset. The TIMIT dataset is relatively small; it provides only few examples for some phones (see Figure 5). After many epochs, our final network started overfitting; it reached an accuracy of roughly 80% on the training set, but only slightly more than 60% on the testing set. This is likely partially caused by data sparseness. The effect of this sparseness can be mitigated to some extent by applying some form of data augmentation to artificially increase the dataset size (Simard et al., 2003; Krizhevsky et al., 2012). For example, phone classes are invariant over small pitch shifts. One of various possible augmentation strategies would be to generate additional phone spectrograms from existing spectrograms by slightly shifting pitches.

3. Many CNN models for speech recognition are not trained on regular FFT-spectrograms, but on, for example, (modified) mel-frequency cepstra. See for example Abdel-hamid et al. (2012) and Tóth (2014). It is conceivable, but unlikely, that results found here do not generalize to other time-frequency representations of audio. However, further research could be done to validate the results found here.

4. The number of frequency bins chosen for the spectrogram generation yields an imperfect round-trip waveform reconstruction. It is possible that this loss of information hampers network classification performance through blurring of phone spectrograms. Furthermore, this has direct negative consequences on the theoretical best waveform reconstruction that can be generated from reconstructed spectrograms. It is possible that the waveform mxcorr would become more nuanced if reconstruction is performed on a network trained on larger spectrograms.

9 Conclusion

This research has looked at the possibility of reconstructing speech input into a Convolutional Neural Network. Furthermore, the reconstructions generated were used to make inferences about the network's learned representation of the input.

I have shown that speech input into CNNs can be reconstructed through using the resulting activation patterns of such networks. To this end, decoding followed by backpropagation provided the best results. Additionally, I have shown that among the measures looked at the maximal waveform cross-correlation between the waveform of the input and the reconstruction provides the best measure of reconstruction quality. Furthermore, this research has probed the internal representations learned by such CNNs by showing (i) that they appear to be insensitive to intensity changes, (ii) that their representations are mostly invariant over scaling in the time-domain, and (iii) that the encoder indeed learns a speech-oriented representation.

References

Abdel-hamid, O., Deng, L., & Yu, D. (2013). Exploring convolutional neural network structures and optimization techniques for speech recognition. In *14th Annual Conference of the International Speech Communication Association* (pp. 3366–3370).

Abdel-hamid, O., Jiang, H., & Penn, G. (2012). Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on* (pp. 4277–4280).

Battiti, R. (1989). Accelerated backpropagation learning: Two optimization methods. *Complex systems*, *3*(4), 331–342.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127. doi: 10.1561/2200000006

Bengio, Y., & Lamblin, P. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, *19*(1), 153—-160. doi: citeulike-article-id:4640046

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010* (pp. 177–186).

Corana, A., Marchesi, M., Martini, C., & Ridella, S. (1987). Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Transactions on Mathematical Software*, *13*(3), 262–280. doi: 10.1145/66888.356281

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Erhan, D., Bengio, Y., & Courville, A. (2010). Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, *11*(2010), 625–660.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., & Zue, V. (1993). *Timit acoustic-phonetic continuous speech corpus LDC93S1.* Web Download. Philadelphia: Linguistic Data Consortium.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statisitics* (Vol. 15, pp. 315–323). doi: 10.1.1 .208.6449

Goffe, W. L., Ferrier, G. D., & Rogers, J. (1994). Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, *60*(1), 65–99. doi: 10.1016/0304 -4076(94)90038-8

Güçlü, U., & van Gerven, M. a. J. (2014). Deep neural networks reveal a gradient in the complexity of neural representations across the brain's ventral visual pathway. *arXiv preprint arXiv:1411.6422*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*. doi: arXiv:1207.0580

Hubel, D. N., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, *160*(1), 106–154.

Kemper, D. (2015). *Understanding the features of a convnet trained for phone recognition* (Unpublished bachelor's thesis). Radboud University Nijmegen.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances In Neural Information Processing Systems* (pp. 1097—-1105).

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323. doi: 10.1109/5.726791

Lee, K.-F., & Hon, H.-W. (1989). Speaker-independent phone recognition using hidden markov models. *Acoustics, Speech, and Signal Processing, IEEE Transactions on*, *37*(11), 1641–1648. doi: 10.1109/29.46546

Lopes, C., & Perdigão, F. (2011). Phone recognition on the TIMIT dataset. *Speech Technologies/Book*, *1*, 285–302.

Ma, Y., Dang, J., & Li, W. (2014). Research on deep neural network's hidden layers in phoneme recognition. *Chinese Spoken Language Processing (ISCSLP), 2014 9th International Symposium on*, 19–23.

Montavon, G., Braun, D. M., & Müller, K.-R. (2010). Layer-wise analysis of deep networks with gaussian kernels. In *Advances in Neural Information Processing Systems* (pp. 1678–1686).

Ng, A. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning* (p. 78).

Riemens, J. M. (2015). *Using convolutional autoencoders to improve classification performance* (Unpublished bachelor's thesis). Radboud University Nijmegen.

Sainath, T. N., Kingsbury, B., Mohamed, A.-r., Dahl, G. E., Saon, G., Soltau, H., ... Ramabhadran, B. (2013). Improvements to deep convolutional neural networks for LVCSR. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on* (pp. 315–320).

Sainath, T. N., Mohamed, A. R., Kingsbury, B., & Ramabhadran, B. (2013). Deep convolutional neural networks for LVCSR. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (pp. 8614–8618). doi: 10.1109/ICASSP.2013.6639347

Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, 958–963. doi: 10.1109/ICDAR.2003 .1227801

Tóth, L. (2014). Combining time-and frequency-domain convolution in convolutional neural network-based phone recognition. *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 190–194.

Vedaldi, A., & Lenc, K. (2014). Matconvnet – con-

volutional neural networks for matlab. *CoRR*, *abs/1412.4564*.

Veselý, K., Karafiát, M., & Grézl, F. (2011). Convolutive bottleneck network features for LVCSR. *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, 42–47. doi: 10.1109/ASRU.2011.6163903

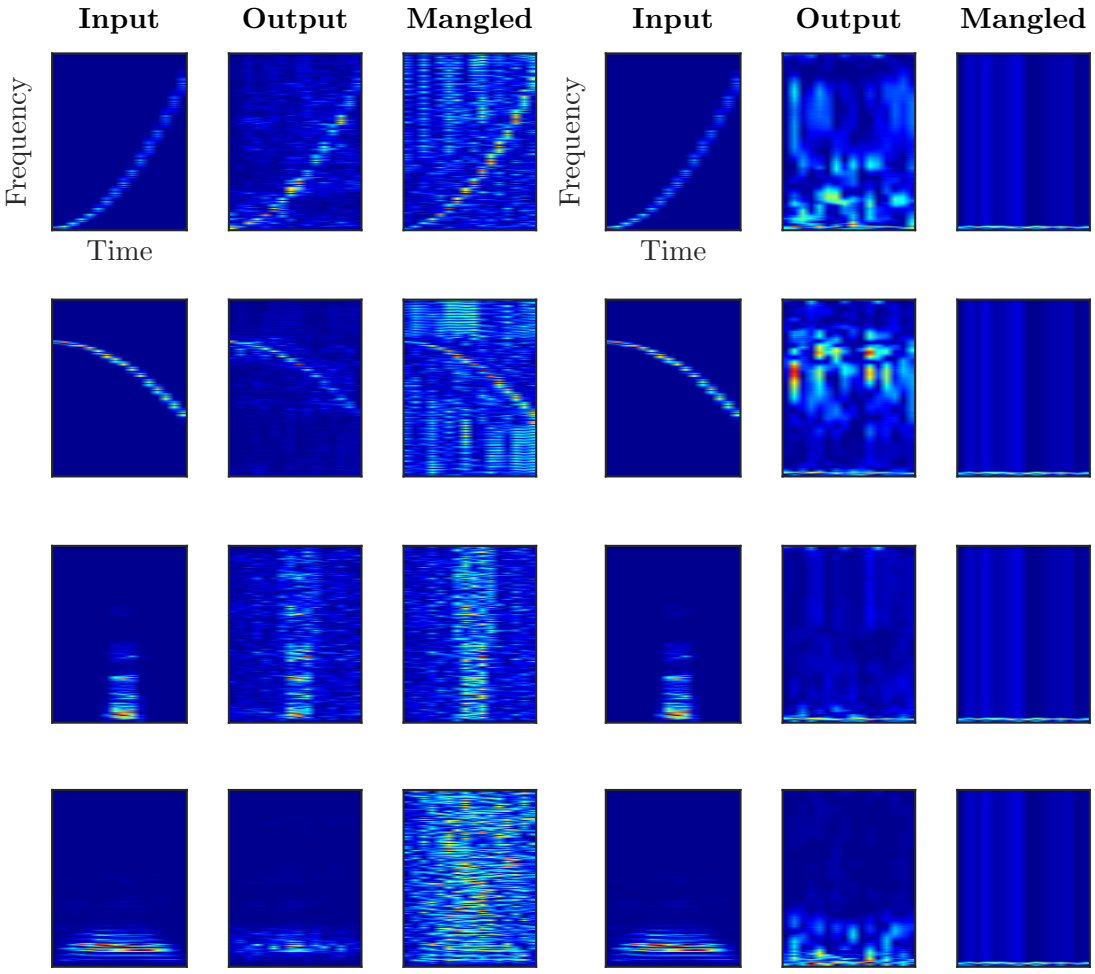
Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, *11*(3), 3371–3408. doi: 10.1111/1467-8535.00290

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*(2), 241–259. doi: 10.1016/S0893-6080(05)80023-1

Zeiler, M., & Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 1–9.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818—-833). doi: 10.1007/978-3-319-10590-1_53

A Unpooling Experiments



(a) The decoded inputs with switch unpooling.

(b) The decoded inputs with blind unpooling.

Figure A.1: Switch-unpooling vs. blind-unpooling. In both figures each row represents the input into the networks, the output generated by the original network and the output generated by the mangled network respectively. The mangled network is generated from the original network by setting the filters and biases of the first convolutional transpose layer to zero.

B Additional Results

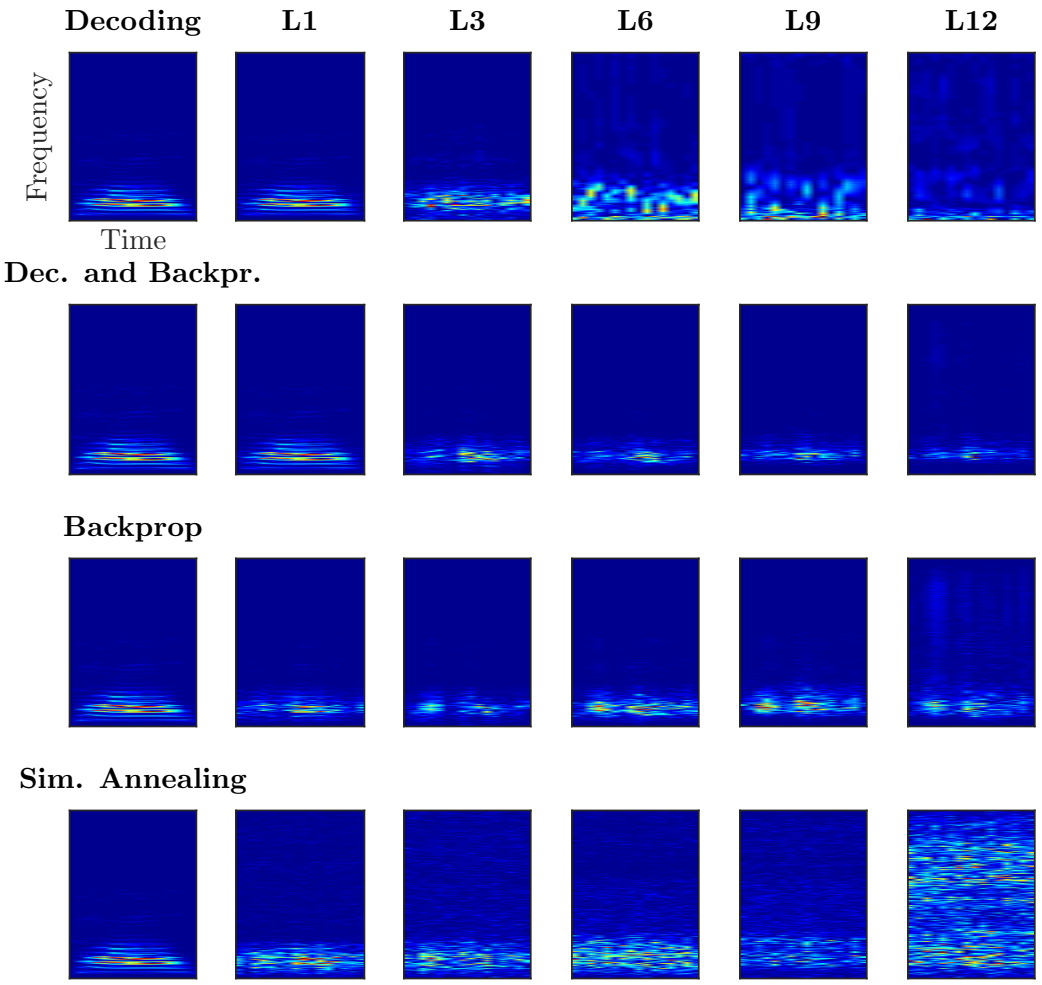


Figure B.1: The generated reconstructions for one case of phone [ɔ]. The different reconstruction methods are shown per row. In the left-most column the original inputs into the network are shown (these are identical over the various methods). The other columns show the reconstructions generated from a certain layer by the corresponding method.

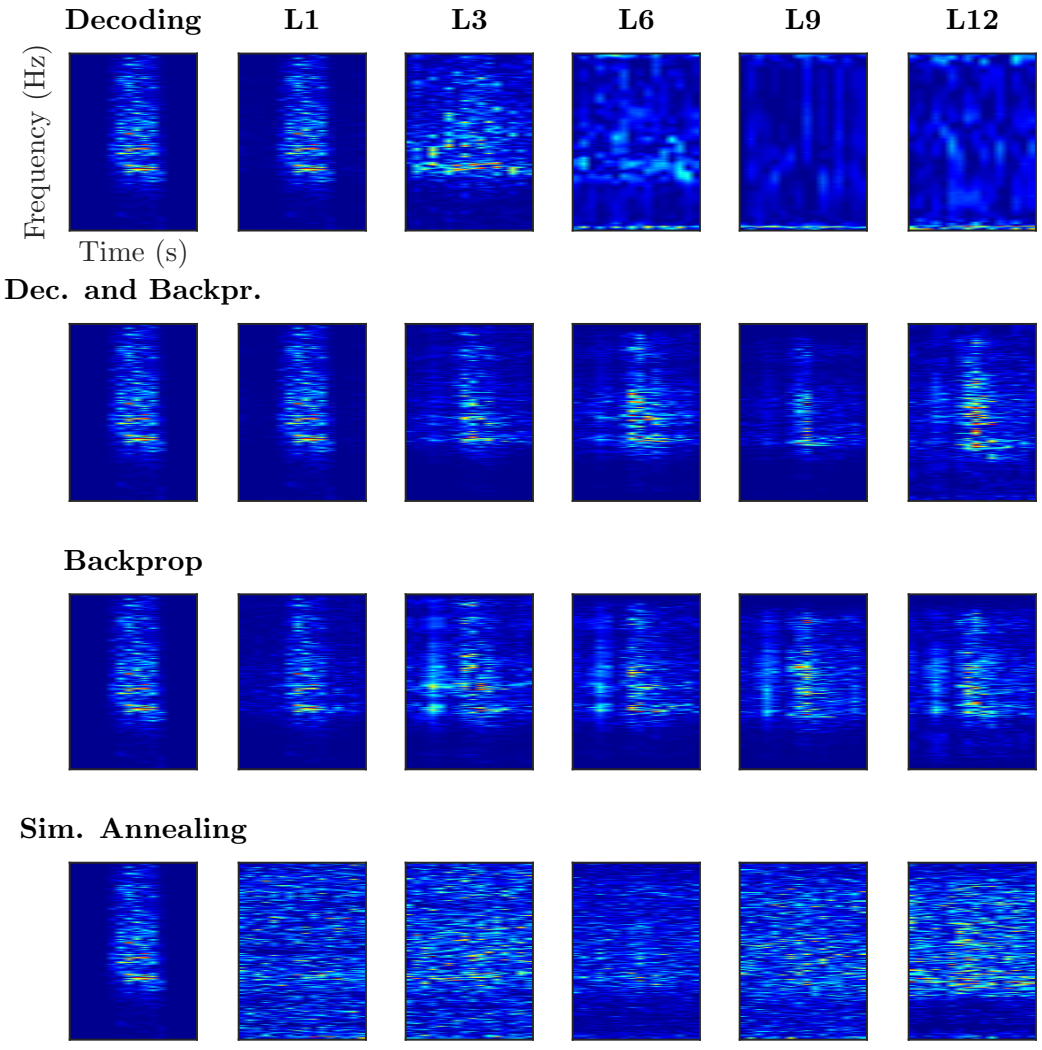


Figure B.2: The generated reconstructions for one case of phone [ʃ]. See Figure B.1 for an explanation.