Radboud University Nijmegen

Bachelor Thesis

in Artificial Intelligence

Understanding the features of a convnet trained for phone recognition

Thesis submitted by:
Diede Kemper
s4056566

Supervisors:
Marcel Van Gerven*
Umut Güçlü*

July 8, 2015

* Donders Institute for Brain, Cognition and Behaviour, Radboud University Nijmegen

Abstract

For convolutional neural networks (convnets) trained for image recognition it is known what the features represent. However, for convnets trained for phone recognition this is not known yet. This study tried to answer the following question: What do the features of such a convnet represent? A convnet with three convolutional layers was trained on the TIMIT phone recognition task and a deconvnet was applied to obtain visualizations of its features. In experiment 1 the deconvnet was applied on the activation caused by the top 4 input phones per feature. In experiment 2 it was applied on the activation caused by the top 3 average phones per feature. Phone label analysis reveals consonant-, front vowel- and back vowel-sensitive features in the third layer. For both experiments, the visualizations were hard to interpret. It could be that visualizing features that represent aspects of audio is not the best way to gain insight into the features, although more experiments that use different convnet architectures should be run to confirm this. Future research could search for other ways to gain insight into the representations of the features, by for example further exploring the possibilities of phone label analysis.

Introduction

Convolutional Neural Networks

Convolutional neural networks (convnets) are a type of deep neural network with three distinctive characteristics: local receptive fields, shared weights and sub-sampling (Lecun et al., 1998). The first layer of the convnet is a convolutional layer, in which each unit is connected to a local receptive field of the input, for example an image. A convolutional layer is composed of multiple feature maps with different weight matrices. Such a feature map is a set of units that all have their receptive fields at different locations on the image, but share the same set of weights. This causes all units in the feature map to be sensitive to the same input, but at different locations on the image. The subsequent layer is a sub-sampling layer. The receptive field of each sub-sampling unit is an area of fixed size in the previous layer. The activation of a sub-sampling unit is either the average (average pooling) or the maximum (max pooling) of the output of the units in its receptive field. The architecture of a convnet is not fixed and thus the number, type and order of the layers can differ between convnets.

Currently, convnets are applied mostly in the field of image classification. Contests in image classification are held to promote research in this field and convnets are true top performers. For example, Krizhevsky et al. (2012) have built a large convnet consisting of five convolutional and max-pooling layers and three fully connected layers and they trained it on the ImageNet dataset which consists of 1.2 million labeled training images. Their model achieved a winning top-5 test error rate of 15.3 percent in the ILSVRC-2012 competition.

Convnets have two properties that cause them to be so good in image recognition. First, compared to standard fully-connected networks, convnets have much less parameters that need to be learned and thus they are easier to train (Krizhevsky et al., 2012). This comes in handy when training on huge datasets such as the ImageNet dataset. Second, convnets are less sensitive to variations in the position of features in the input images than regular feedforward neural networks (LeCun & Bengio, 1995). This is caused by the weight sharing property of convnets. Because weights are shared, units will be sensitive to the same features, and because their receptive fields are at different locations on the input they can recognize the same feature at all locations. For example, the network of Krizhevsky et al. (2012) could correctly classify a mite even when it was depicted at the edge of the image. Third, whereas fully-connected networks ignore the topology of the input, the local receptive fields of the units of convnets force the network to

extract and combine local features before recognizing the full image (LeCun & Bengio, 1995).

Visualizing the features

Visualizing the features that the feature maps of a convnet represent can be a good way to gain insight in the network. When visualizing a feature of a network, the activation of the corresponding feature map is projected back onto input space. In other words, it is decided which input activation could probably have caused the activation within the feature map and this input activation is visualized. Visualizing the features of the first layer is easy, because the first layer is connected directly to input space. However, for features of deeper layers visualization is hard. Zeiler & Fergus (2014) introduced a new technique that can be used to visualize the features of these deeper layers. They used a Deconvolutional Neural Network (deconvnet), which can be seen as the opposite of a convnet. A convnet maps pixels to features and a deconvnet maps features to pixels.

Zeiler & Fergus (2014) trained a convnet with the same architecture as the award winning network of Krizhevsky et al. (2012). They trained the network on the ImageNet dataset and then visualized its layers, which showed two important results. First, visualizations of deep layer features were more complex than those of first layer features. In layer 1, stripes and borders were represented. In layer 2, corners and circles were visible. In layer 3 and 4 the features became more complex, up to layer 5 where whole objects such as faces and petals were represented. Second, Zeiler & Fergus (2014) showed that visualizing the layers of a convnet can help finetune the architecture of the network. For example, the authors found that the first layer filters focused too strongly on the high and low frequency information and not enough on the mid frequencies. Moreover, the visualization of the second layer showed aliasing artefacts. After finetuning the network parameters to remedy these problems, the new network performed better in terms of classification accuracy.

Application to audio

The application of convnets for speech recognition is a new emerging field. In 2012, Hinton et al. wrote an overview paper about the application of deep neural networks to speech recognition, with only one paragraph dedicated to the application of convnets to speech recognition. One year later, the same research teams as before wrote an extra overview paper to describe the advances of that year. One of the new discoveries they mention was that convnets work better than plain deep neural networks because they have less parameters to train and the pooling gives some invariance to vocal tract differences between speakers (Deng et al., 2013).

There has not been much research on the mechanisms by which convnets trained for speech recognition exactly work. One of the only studies in this matter was conducted by Ma et al. (2014). They have studied the physical meaning of the hidden layers of a deep neural network trained for speech recognition. They first trained a seven-layered deep neural network on the TIMIT phone recognition task and then compared its performance with the performance of the network when one of its layers is removed. The idea behind this method was that the performance of the network with one removed layer tells us something about the responsibility of that layer. When the probability of correctly classifying a certain phone decreases when a layer is removed from the network, it is probable that the removed layer is responsible for recognizing that type of phone. However, when the probability of correctly classifying a phone does not decrease when removing a layer, it is probable that the layer is not responsible for recognizing that type of phone.

Using this method, the authors found that the first layers are responsible for the back vowels[1],

[1]Back vowels are vowels produced by changing the position of the back of the tongue, for example the 'a' in 'car'. Front vowels are produced by changing the position of the front of the tongue, for example the 'i'

whereas the latter layers are responsible for the front vowels. Moreover, they found that the first hidden layer is responsible for most of the consonants with the constriction located in the front of the vocal tract[2], whereas the other consonants are processed by the middle and higher layers. Why this pattern arises is yet unknown.

Research questions

There is much to discover about the mechanisms by which convnets trained for speech recognition exactly work. This study will dive deeper into this, by answering the following research questions: what do the features of a convnet represent when the convnet is trained to classify phones? And how do the features change from layer to layer? To answer these questions, a convnet is trained to classify phones and a deconvnet is applied to the activation of its feature maps. Then, in order to interpret the result, the output of the deconvnet is visualized.

There are two main hypotheses about the result of this study. The first hypothesis is that features will get more complex when moving from the first to the latter layers, just like Zeiler & Fergus (2014) found when visualizing the features of a convnet trained for image recognition. The second hypothesis is that features will represent those parts of a spectrogram that give information about the phone that is depicted. But what are these relevant parts of spectrograms? That question is answered within the field of acoustic phonetics. Here, a short summary of the answer will be given, mainly based on the book by Ladefoged & Johnson (2011).

In spectrograms, vowels can be recognized by looking at the formants. Formants are peaks of energy at particular frequencies (Harley, 2008). The two lowest formants distinguish vowels from each other. The lowest formant F1 is related to the 'height' of a vowel, which is whether the tongue moves up or down to produce the vowel. The distance between formants F1 and F2 is related to the 'backness' of a vowel, which is the extend to which the vowel is produced in the front or the back of the mouth. When the frequency of formants change over time, this indicates that the shape of the mouth has changed over time and that the phone is a diphthong: a combination between two other vowel sounds. Figure 1 shows the spectrograms of a vowel and a diphthong[3].

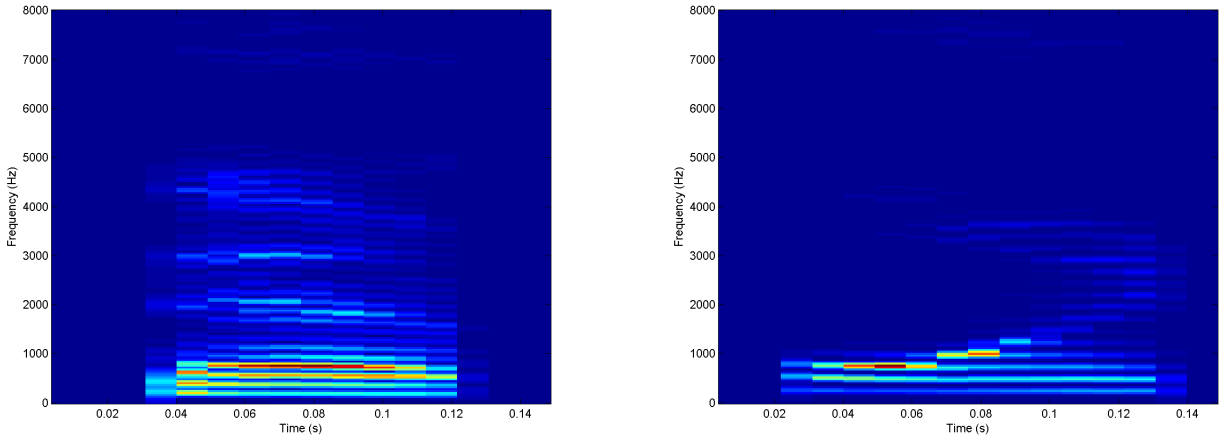
Consonants often show more complex spectrograms than vowels. Consonants can be seen as particular ways of beginning or ending a vowel and do not always have distinguishing characteristics themselves. Consonants can be distinguished by looking at attributes such as the onset and offset frequency of each formant, whether the formant changes in frequency, whether there is a sudden or gradual silence, whether there is noise and if so, at which frequency and the intensity of the different frequencies. Figure 2 shows two spectrograms of consonants. The first shows the nasal consonant 'n'. Nasal consonants start abrupt. Their formant structure is similar to that of a vowel, except that the formants are fainter. There is usually a very low first formant centered at about 250 Hz. Different nasal consonants can be distinguished by looking at the formant transition patterns at the onset and end of the phone. The second spectrogram in Figure 2 shows the consonant 's', which is a voiceless fricative. Voiceless fricatives can be recognized by random-like noise in higher frequencies. For the 's' in particular this noise has a very high intensity.

To summarize, based on the previous section, it is hypothesized that the features in the first layers represent formants, their onset and end points and random noise patterns. Furthermore, it is hypothesized that in latter layers the position of noise and

in 'bit'.

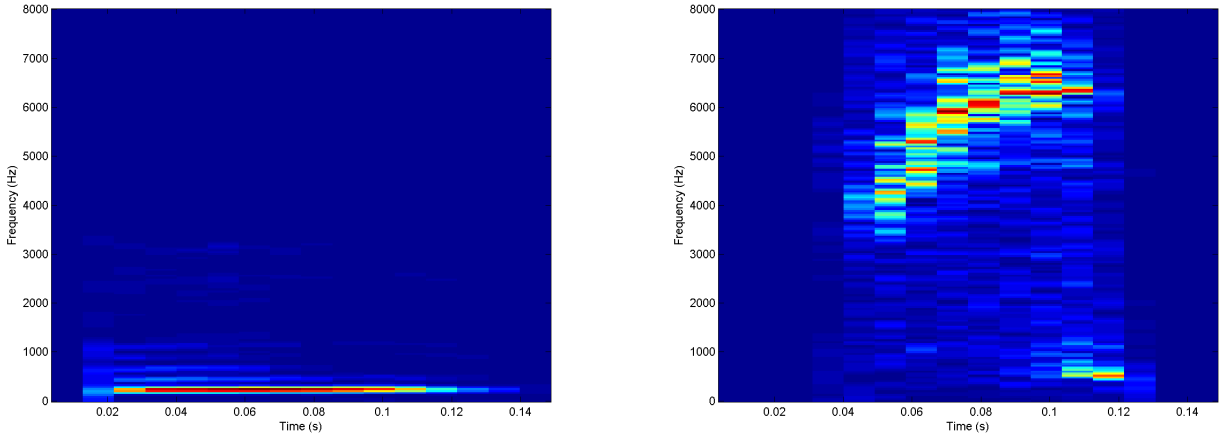
[2]An example of a consonant with the constriction located in the front of the vocal tract is the 'p' in 'pie'. A consonant with the constriction located more deeply in the vocal tract is the 'k' from 'kid'.

[3]In this thesis the same phone labels are used as in the TIMIT database. The book chapter by Lopes & Perdigao (2011) is used to know which phone label matches which sound.



(a) 'eh' (b) 'oy'

Figure 1. Panel a) shows the spectrogram of the vowel 'eh', for example the 'e' in 'bed'. The horizontal bars of high energy are the formants. Panel b) shows the spectrogram of the diphthong 'oy', for example the 'oy' in 'boy'. It can be seen that the formants change frequency over time.



(a) 'n' (b) 's'

Figure 2. Panel a) shows the spectrogram of the nasal consonant 'n', for example the 'n's in 'noon'. Panel b) shows the spectrogram of the consonant 's', for example the 's' in 'sea'.

formants relative to each other and their frequencies are represented. Besides the two hypotheses it will also be checked whether the features between the layers differ in some way that could explain the results found by Ma et al. (2014).

Materials and Methods

In this section, the materials and methods that are used in this study are described. Some of the work described in this section was performed together with Riemens (2015) and Churchman (2015). To be precise, the preprocessing of the auditory data and the design and training of the

convnet were shared activities. All other work that is described was performed by the author of this thesis herself.

Preprocessing the auditory data

The dataset that is used in this study is the TIMIT dataset (Garofolo, 1993). TIMIT contains the recordings of 630 speakers each reading out loud ten sentences in American English. Two sentences are read by all speakers, the other eight sentences vary between speakers. The dataset is divided into a balanced train and test set. The dataset was read using the Matlab Audio Database Toolbox, that automatically splits the audio data in slices that represent phones. The sentences that were read by all speakers (with codes 'SA1' and 'SA2') were not included to prevent some phones to be overrepresented, because this could bias the results (Abdel-Hamid et al., 2012). Following Lee & Hon (1988), the 61 phone labels were converted to a set of 48 phone labels, such that similar phones are grouped together under the same label.

The inputs to the convnet must be of equal size. Therefore, the audio data of each phone was padded with zeros on both sides such that it was as long as the longest phone in the dataset. However, to save memory, the largest 5 percent of the phones were excluded before zero padding the data. Then, the data of each phone was represented as a spectrogram, using the short-time fourier transform[4]. This was done using a 16ms Hanning window and a fixed frame rate of about 9ms. The number of frequency bins (y-axis) was 201 for each spectrogram[5]. Moreover, all spectrograms (of both training and test phones) were normalized by subtracting the average of all training spectrograms. At the end, each of the 126691 train and 45744 test phones was represented as a spectrogram with 201 frequency bins and 16 time windows.

Convnet architecture and training

In the appendix it is explained in detail how deep neural networks, and in specific convnets, work and how they can be trained. For the implementation of the convnet the Matlab toolbox MatConvnet (Vedaldi & Lenc, 2014) was used. The code that was used to train the network was based on example code available in the toolbox. For training, stochastic gradient descent was used, which was already implemented in the code, with a batchsize of 128 phones per batch. To make the training faster, the network was trained on a GPU. Moreover, batch loading and network training were done on two separate threads. To make multithreading possible, some of the code had to be rewritten to C++. As can be seen in Figure 3, the number of train examples per phone class differs tremendously. To solve this problem, (re)sampling took place every epoch such that there was a uniform distribution of classes[6]. The network was trained for 372 epochs. Adagrad was implemented to make the learning rate annealing automated (Dyer, n.d.). Moreover, a dropout layer with a dropout rate of 0.9 was added to the network to combat overfitting. Also, following Lee

[4]Two other types of spectrograms have been considered. For the first type, the gammatone based spectrogram, the problem was that data took too much memory when stored in this form. With the second type, the melscale based spectrogram, the convnet did not perform as well as with the short-time Fourier transform spectrogram type.

[5]This parameter was chosen such that there was a nice trade-off between network training speed and reconstruction quality, with reconstruction quality measured as the correlation between the original signal and the signal that arises from inverting the spectrogram. With 201 frequency bins, this correlation was 0.91. For comparison, with 101 frequency bins this correlation was 0.36 and with 301 frequency bins this correlation was 0.98.

[6]Sampling took place such that each class had 500 train examples per epoch. In case the class did not have 500 different train examples, all train examples were used at least once and some train examples were resampled (without replacement) such that the total number of examples was 500. The same method was used such that each class had 200 test examples per epoch.

& Hon (1988), the convnet was trained to classify 48 phone classes, but validated with only 39 phone classes, such that confusions between some of the 48 phone classes were not seen as errors.

Multiple different architectures of convnets have been considered. Their performance was compared by training each convnet for 10 epochs on a section of the training data. The first main architecture that was considered used convolution across both axes. It did not perform well and changing parameters did not add much to its performance. The second main architecture that was considered used convolution across the time axis only, inspired by the time-delay neural networks (Waibel et al., 1989). It performed quite well: with two convolutional layers and two fully connected layers it had an phone recognition accuracy of 48.7%. The third main architecture that was considered used convolution across the frequency axis only. This architecture was inspired by the paper of Abdel-Hamid et al. (2013), who showed that convolution across frequency axis works better than convolution across time axis for their convnet. As a possible reason for the difference in performance the authors mention that the input to their convnet only has 15 time windows compared to 40 frequency bins, and thus convolution across frequency has more benefit because there is more to convolve over. In the current study, this is also the case: The input has 16 time windows compared to 201 frequency bins, and thus convolving across frequency makes more sense. Using this architecture with two convolutional layers and two fully connected layers, the convnet had a phone recognition accuracy of 59.1%.

The final version of the convnet scored a phone recognition accuracy of 62.6% and its architecture is as follows. All units, except for the units in the last layer, are Rectified Linear Units, which means that their activation function is $f(x) = \max(0, x)$. The convnet starts with three convolutional and max pooling pairs. The first convolutional layer consists of 8 units each with a filtersize of 8 frequency bins and 16 time windows. This filtersize covers all time windows and thus the filter only convolves across the frequency axis. The second dimension of the filters of the second and third convolutional layer is of size 1, because the time axis is reduced to size 1 in the first convolutional layer and thus the filters cannot be larger. The second convolutional layer consists of 16 units, each with a filtersize of 6×1. The third convolutional layer consists of 32 units each with a filtersize of 4×1. Each max pooling layer has a pool size of 5×1 and a stride of 2. The last max pooling layer is followed by three fully connected layers, each consisting of 1000 units. The convnet ends with one fully connected layer consisting of 48 units. This last layer has the softmax function as its activation function, which causes the output of the network to be a categorical probability distribution.

Visualizing features by using a deconvnet

Zeiler & Fergus (2014) have introduced the deconvnet as a new technique that can be used to visualize the features of a convnet. The following section describes how this technique works. As described before, a deconvnet can be seen as the opposite of a convnet. For each layer in the convnet the deconvnet has a corresponding layer that approximately inverts the operation of the convnet layer: Each max pooling layer has a corresponding max unpooling layer and each convolutional layer has a corresponding deconvolutional layer. Figure 4 shows how a deconvnet (on the right) is attached to a convnet (on the left) to be able to visualize the features of the convnet.

Suppose the goal is to visualize a certain feature F within the second convolutional layer of the convnet. The first step is to select the input for the convnet that will cause the most activation within the feature map of F. Let us call the location of this maximum activation within the feature map location L. Then, a forward pass is performed through the convnet with this input (the grey arrows that go up in Figure 4) up to the second convolutional layer. The activation in this last layer is copied to the corresponding deconvolutional layer in the

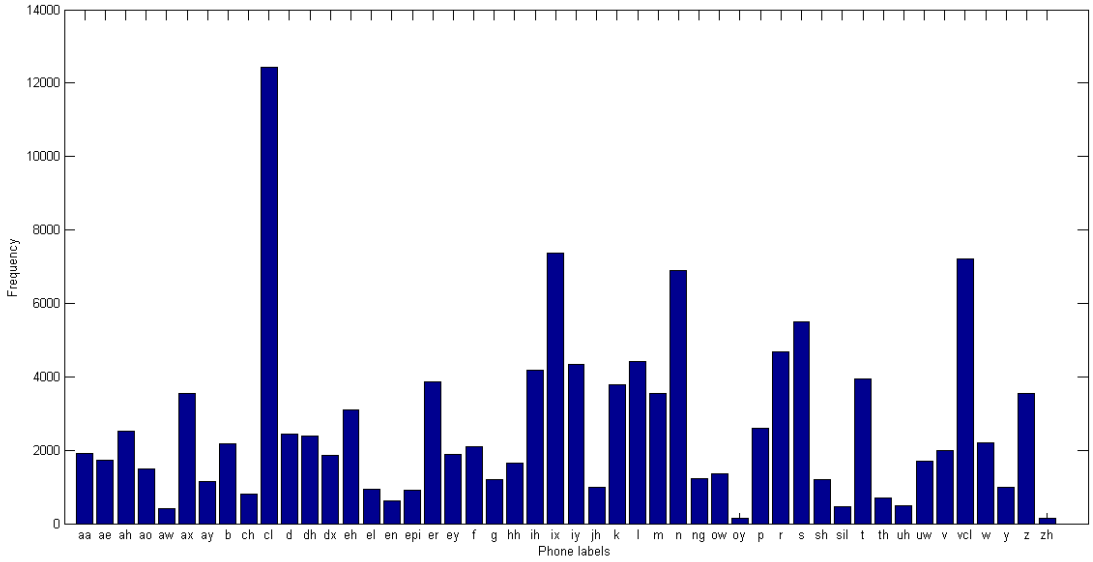


Figure 3. The number of train examples per phone class (folded to 48 classes).

deconvnet. Then, all activation in the deconvolutional layer is set to zero, except for the activation at location L within the feature map of F. Again, a forward pass is performed, this time through the deconvnet (the grey arrows that go down). Note that the layers of the deconvnet have the opposite order as the layers of the convnet. The output of the deconvnet shows the part of the input that has activated location L. If this output is visualized, it shows to what input feature F is sensitive.

A deconvnet has two types of special layers. The

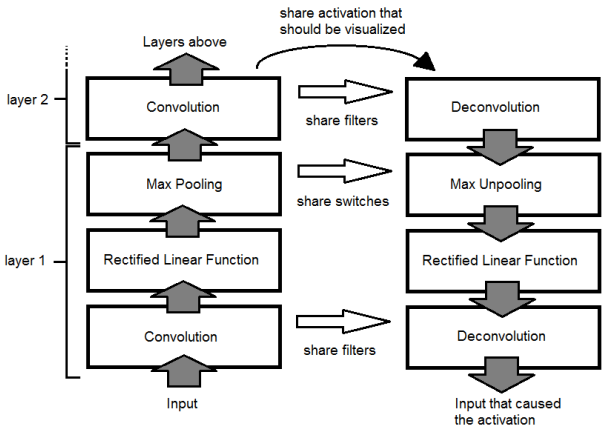


Figure 4. A convnet (left) with a deconvnet (right) attached to it.

first is the max unpooling layer, which approximately inverts the operation of the max pooling layer of the convnet. In Figure 5, both the max pooling and max unpooling operation is depicted. As can be seen, a max pooling layer takes the maximum value from, in this case, a 2×2 pooling area as the value of the units. It also saves the location of the maximum value for each pooling area in the so called switches. The max unpooling layer takes the maximum values and places them back onto the locations where they came from. The layer uses the switches to know at what locations the maximum values should be placed. The second special layer of the deconvnet is the deconvolutional layer, which approximately inverts the operation of the convolutional layer of the convnet. The deconvolutional layer uses the same filters as the corresponding convolutional layer, with the only difference that the filters are flipped both vertically and horizontally. It then convolves these filters over its zeropadded input.

Springenberg et al. (2015) have extended the deconvnet technique by Zeiler & Fergus (2014) by extending the way the rectified linear function is approximately inverted, as shown in Figure 6. During the forward pass, the rectified linear function

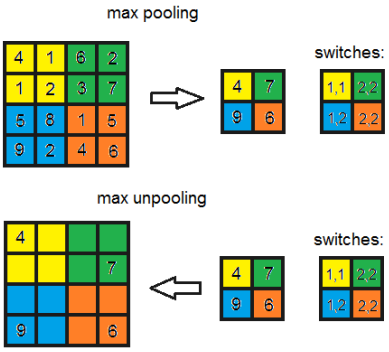


Figure 5. In the max pooling layer, the location of the maximum values are stored in the so called switches. The max unpooling layer uses these switches to know where to put the maximum values.

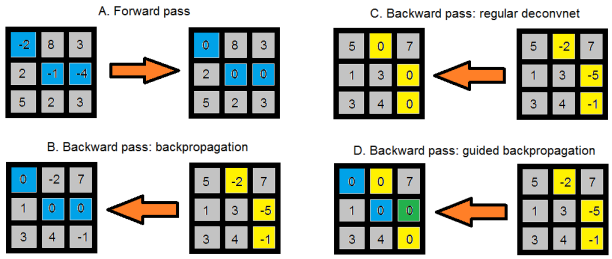


Figure 6. Panel A shows the forward pass, in which the rectified linear function is applied to all values. Panels B to D show different ways to execute the corresponding backward pass. Panel B shows the backward pass during backpropagation and Panel C shows the backward pass in a deconvnet. Panel D shows the method of guided backpropagation, which combines the methods that are used during backpropagation and in a deconvnet.

sets all negative activation to zero (Panel 6A). During backpropagation and in a deconvnet, the backward pass through this function is executed in a different way. During backpropagation, all values that were set to zero during the forward pass are set to zero in the backward pass too (Panel 6B). In a regular deconvnet, the rectified linear function that is applied within the convnet is approximately inverted by applying the same function within the deconvnet (Panel 6C). Springenberg et al. (2015) found that using the guided backpropagation method, which combines the regular deconvnet method with the method used during backpropagation, gave much less noise in the results of the deconvolution process, mainly when visualizing the features in higher layers.

The current study

The current study consisted of two experiments. In the first experiment, the features of the three convolutional layers were visualized using a deconvnet with the extra method of guided backpropagation. This was implemented in Matlab[7]. For each feature, the 4 input spectrograms within the test set that caused the most activation in the feature map were selected. These 4 phones were then used to visualize the aspects of the phone that the feature was sensitive to.[8]

The results of the first experiment were hard to interpret, because the input phones used in the deconvolution process of different features were very similar and thus did not convey much information about the features. Therefore, in the second experiment the deconvolution method was used in a different manner. First, for each of the 48 phones within TIMIT an average phone was created based on the spectrograms within the test set. The intensities of the average phones were set to vary between 0 and 1, to ensure that a difference in intensity could not cause all features to be sensitive to the same phones. Then, for each of these average phones the maximum activity of each feature was computed. The top 3 average phones per feature were then used to visualize the aspects of the phone that the feature is sensitive to, using the

[7]Code for deconvolution was retrieved from the following url (2015-07-04): `https://github.com/umuguc/matconvnet/tree/deconvnet/examples/deconvnet`.

[8]Seven feature maps of the third convolutional layer did not have a positive activation with any of the input phones. The corresponding features were not visualized and excluded from further analysis.

same method as in experiment 1. The results of the two experiments will now be described.

Results

Experiment 1

Figures 7, 8 and 9 show some of the visualized features of respectively the first, second and third convolutional layer[9]. The intensity of all projections was changed such that it ranged from 0 to 6 within each feature. Figure 10 shows the projection of a feature of the first layer. As can be seen, the projection does not cover the whole input space. This was true for all features of the three convolutional layers. Therefore, only the activated parts were visualized. Note that the exact frequencies at which the projections appeared is not important: all features are convolved across the frequency axis and can react to activity across the full input space. As with the projection in Figure 10, almost all projections appeared in the lower frequencies. Reason for this is that the lower frequencies have a higher intensity in human speech (Ladefoged & Johnson, 2011), and therefore lower frequencies will be able to activate a feature more strongly than higher frequencies. In the following paragraphs, we will take a closer look at the features layer by layer.

Figure 7 shows the features of the first convolutional layer. For every feature only the projection of the maximum activation is shown, because all top 4 activations showed a very similar projection. As can be seen, the features show diverse patterns of sensitivity over time. Some features are sensitive to activity at the onset of the phone, whereas others are sensitive to activity at the end or at the middle of the phone. Some features are sensitive to a very short burst of activity whereas others are sensitive to a more noisy pattern of activity.

Figure 8 shows six out of sixteen features of the second convolutional layer. For each feature the projections of the top 4 activations are shown. Note that each projection covers two to three times more frequency bins as the features of the first layer. The features look quite noisy, but some patterns can be distilled: again, some features are sensitive to activation mainly at the onset (feature (a)) or at the end of the phone (feature (b)). Also, some features are sensitive to high activation in the middle of the phone (feature (c)) whereas other features are more sensitive to activation at both the onset and the end of the phone (feature (d)). However, in most features it is hard to see clear patterns.

Figure 9 shows six out of 32 features of the third convolutional layer. For each feature the projections of the top 4 activations are shown. Note that each projection covers twice as much frequency bins as the features of the second layer. As with the features of the second layer, it is difficult to see patterns clearly by eye. It is visible that sometimes activation is more at the onset and sometimes more at the end of the phone. Moreover, the features differ in how wide their activation is. For example, feature (a) is sensitive to a thinner stroke of activation than feature (d). Another thing that draws attention is that feature (c) looks like it represents two horizontal lines of activation at the right side of the feature. When looking at the input phones that were used for these projections, three out of four phones indeed show two clear formants (see Figure 11).

Table 1 shows the similarities of the different features quantified by two measures. The first measure is the sum of squared errors, which for each datapoint takes the squared error between both features and then sums these. Thus, the more similar two features are, the lower their sum of squared errors is. The second measure is the average column-wise cross correlation. For each time point in the features, the normalized cross correlation is taken and the maximum value of the created sequence is computed. Then, the average of the maxima of all time points is computed. If two features are very similar, their average column-wise cross correlation is very close to one.

[9]Note that only a part of the features is depicted because of shortage of space. All results that are not depicted are send along with this thesis.

These two measures are used to compute two similarity scores. First, the between-feature similarity, which is the similarity of projections of different features. Second, the within-feature similarity, which is the similarity of projections of the same feature. As can be seen in the table, from both measures the same conclusions can be drawn. To begin with, for the first layer the average within-feature similarity of the features is very high, whereas the average between-feature similarity is lower. Furthermore, for the features of the second and third layer, the average within-feature similarity is only a bit higher than the average between-feature similarity.

Figure 12 shows how often each phone label is used as a top 4 input phone for the deconvolution process. As can be seen, vowels are over-represented. Almost all input phones used for deconvolution are vowels. Thus, the labels of the top 4 input phones do not convey much information about the features. Moreover, the input spectograms are very similar and often do not show any clear feature-specific characteristics. There are two exceptions. First, the input phones of one feature of the second convolutional layer (feature (e) in Figure 8) are all very short. Figure 13 shows the spectrograms of its four input phones. As can be seen, all phone labels are different, but the four phones are all very short. Thus in some way, the feature is sensitive to a pattern common in short phones. Second, the input phones of one feature of the third convolutional layer (feature (f) in Figure 9) are all consonants, with three out of four being s-like sounds. Figure 14 shows the spectrograms of its four input phones.

There are two possible explanations for why almost all input phones used for deconvolution are vowels. First, vowels show more intensity on average and therefore could cause more activation within features. Figure 15 plots for each phone label the average maximal intensity of its spectrograms. As can be seen, the spectrograms of vowels tend to have a higher maximum intensity than the spectrograms of consonants. Second, vowels show very clear formants that are often quite constant in frequency over time. These formants show in spectrograms as horizontal lines of high intensity. Since all features are thin and horizontal, it could be that they 'catch' most activation when on top of a formant.

Thus, it is not strange that the input phones used for deconvolution were mostly vowels. It does make it difficult to interpret the results. Therefore, a second experiment was run, that focussed more on making clear to which phones each feature is most sensitive.

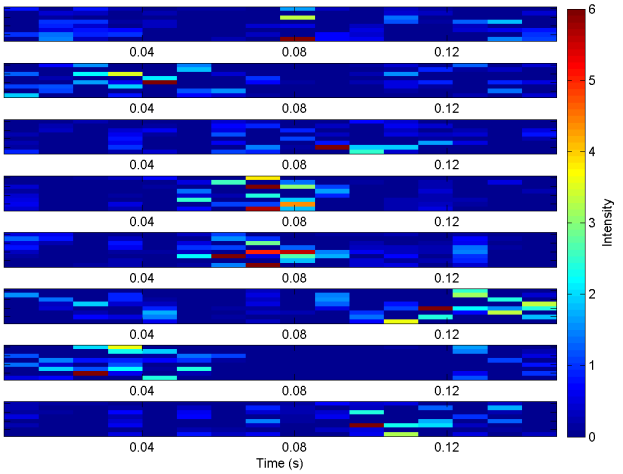


Figure 7. All features of the first convolutional layer. For every projection, the y-axis covers 8 frequency bins which is equal to a range of about 300 Hz.

Table 1
Per layer, the average between-feature and within-feature similarity scores for the projections created in experiment 1. Similarity is measured as the sum of squared errors (SSE) and as the average column-wise cross correlation (corr).

	Between-feature		Within-feature	
Layer	SSE	corr	SSE	corr
1	14.3590	0.4355	<0.0001	1.0000
2	31.9019	0.5295	28.2896	0.5992
3	45.9228	0.5064	44.2928	0.5249

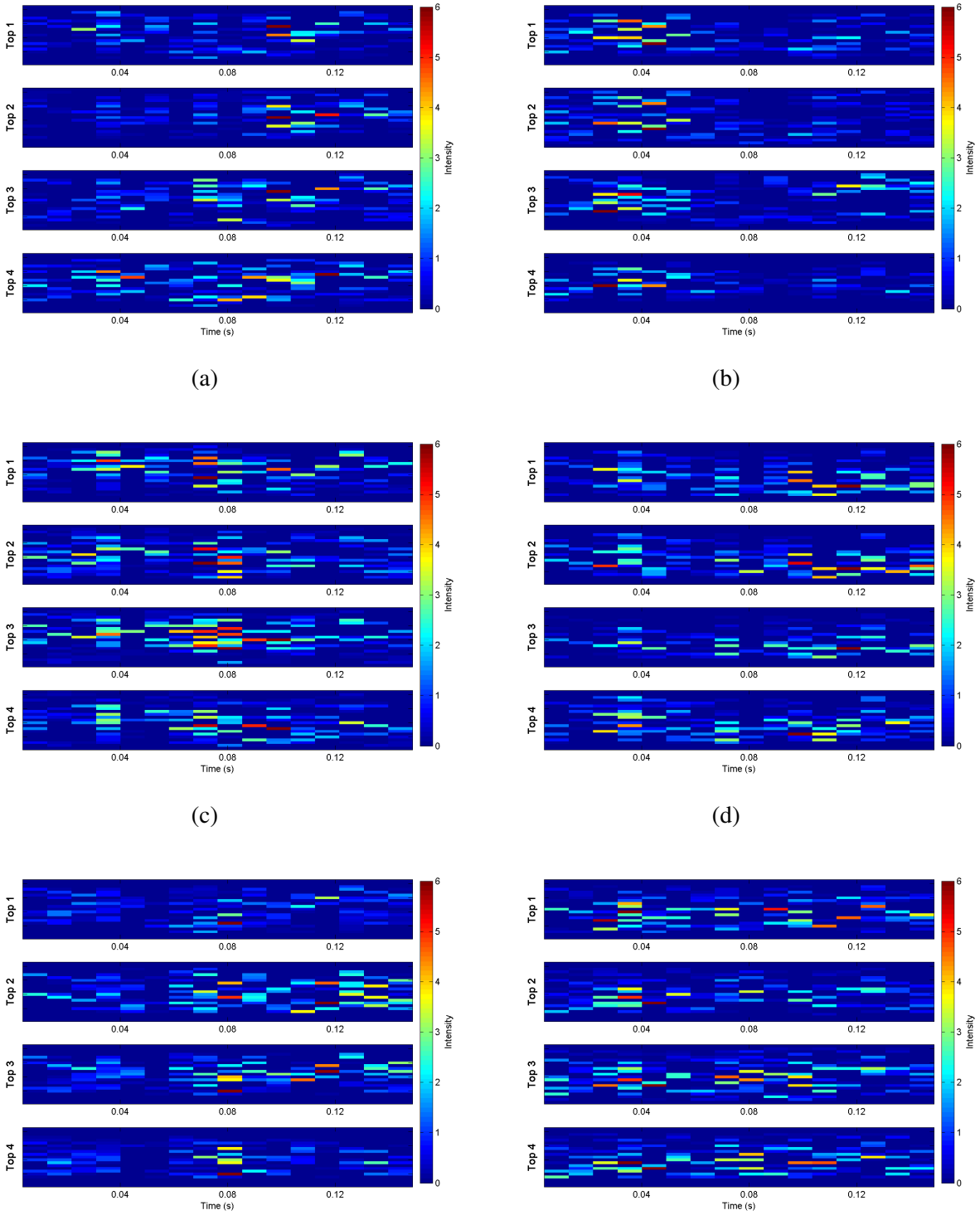


Figure 8. Six out of sixteen features of the second convolutional layer. For each feature the projections of the top 4 activations are shown. For every projection, the y-axis covers 21 frequency bins which is equal to a range of about 850 Hz.

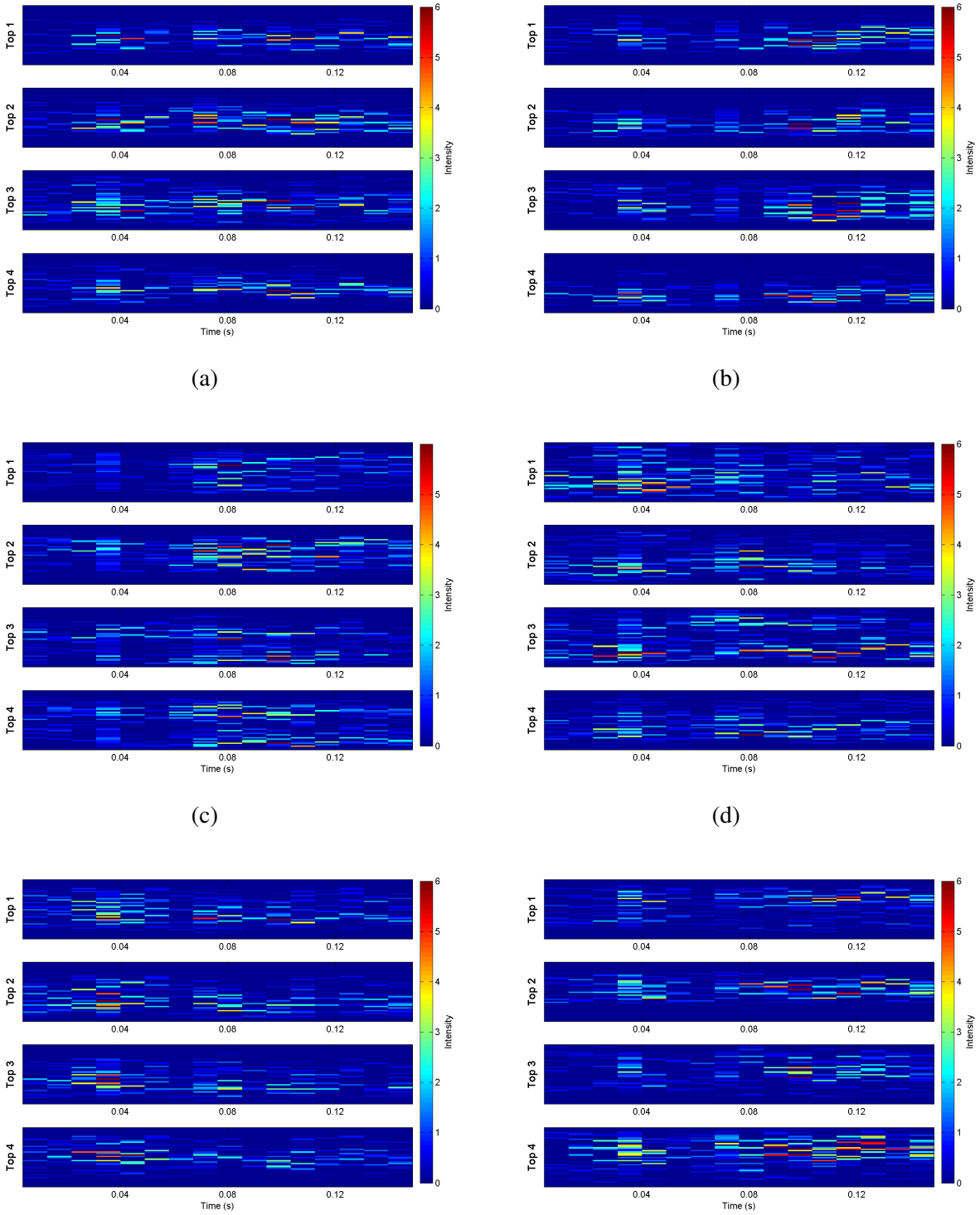


Figure 9. Six out of 32 features of the third convolutional layer. For each feature the projections of the top 4 activations are shown. For every projection, the y-axis covers 42 frequency bins which is equal to a range of about 1700 Hz.

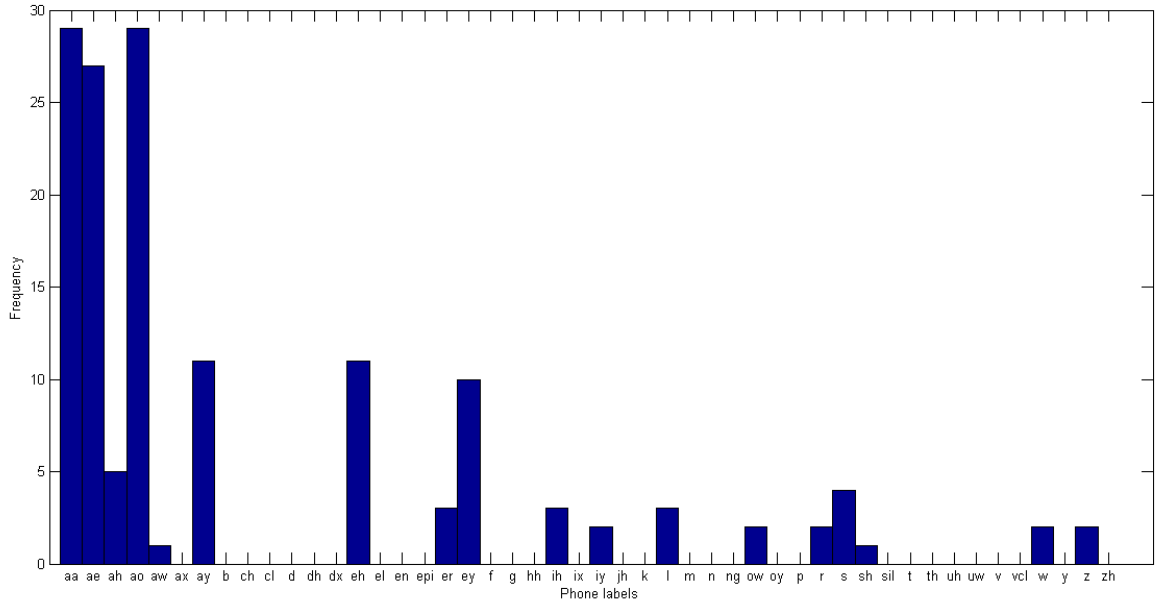


Figure 12. Histogram that plots how often each phone label appeared in the top 3 input phones of features (experiment 1).

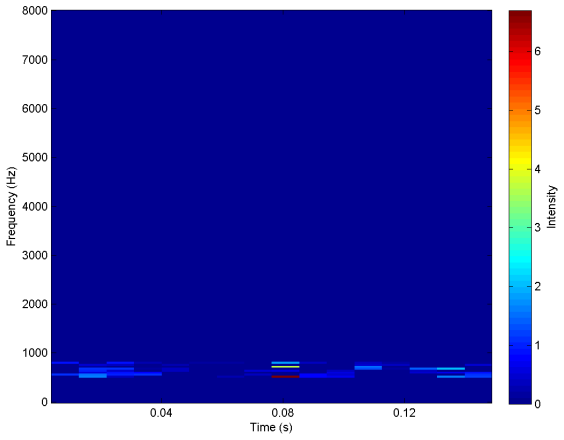


Figure 10. The full projection of a feature of the first convolutional layer.

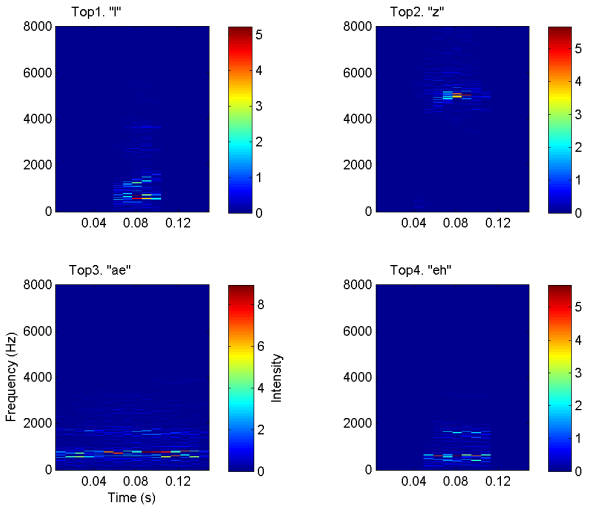


Figure 11. The top 4 inputs of feature (c) of the third convolutional layer.

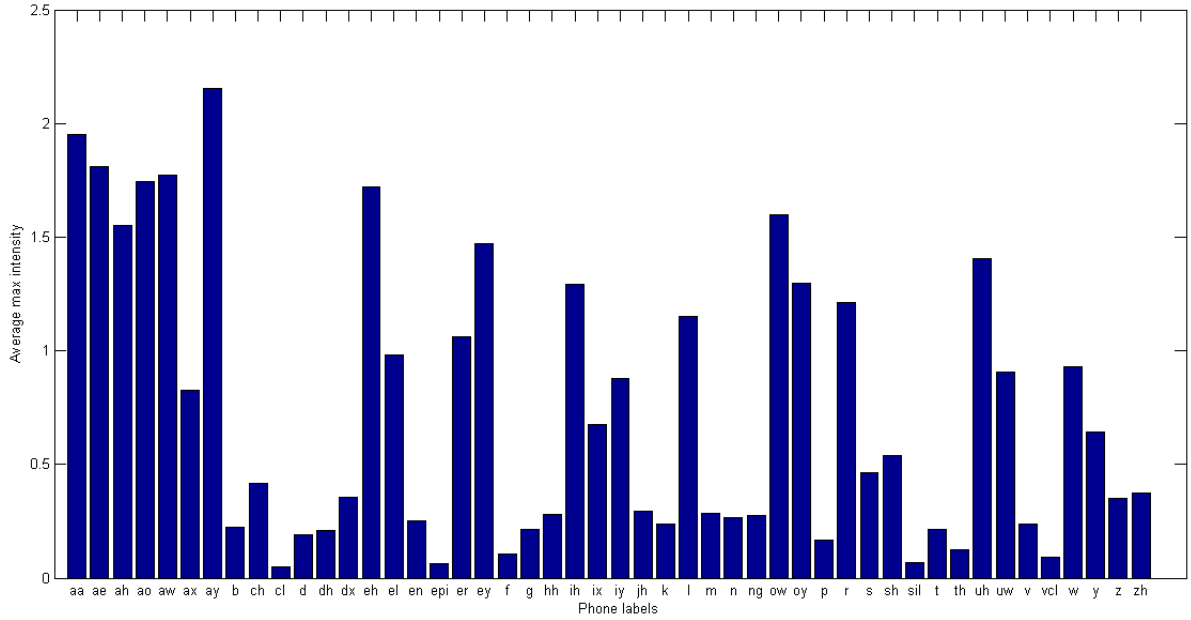


Figure 15. Per phone label the average maximal intensity of its spectrograms.

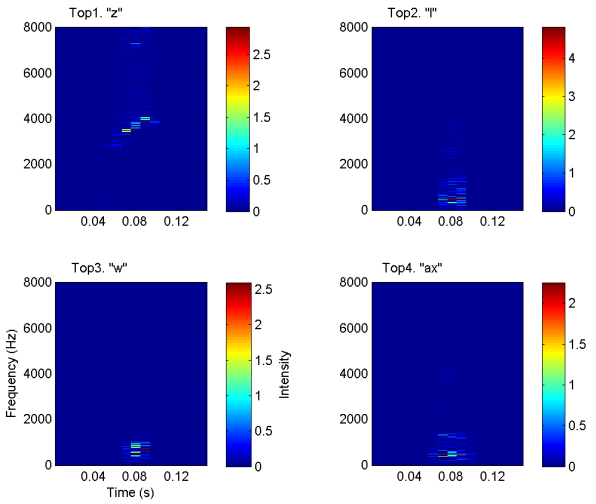


Figure 13. The top 4 inputs of feature (e) of the second convolutional layer.

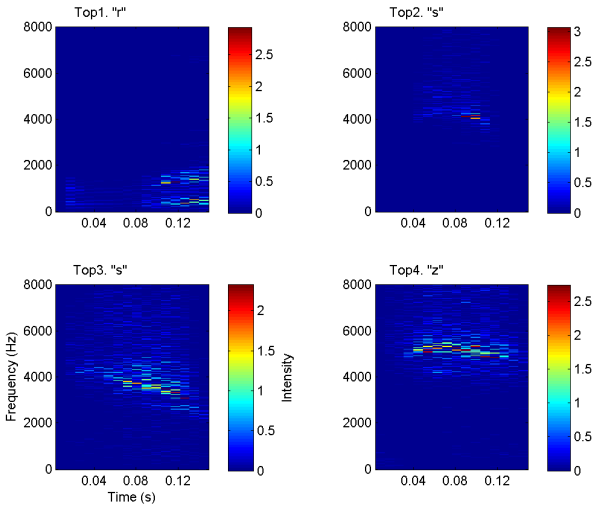


Figure 14. The top 4 inputs of feature (f) of the third convolutional layer.

Experiment 2

The results of this second experiment is divided in two sections. First, the actual projections of the features are analysed. Second, the phone labels that were used for these projections are analysed.

Projections. Figures 16, 17 and 18 show some of the projections of the features[10]. Visual inspection of the projections themselves does not result in more information than we already had: besides the sensitivity to high activation in different points in time it is hard to notice any clear patterns. A difference with the results of the first experiment is that there is more variation in the location of the projection in the spectrogram. For example, Panel 17b) shows a feature of which the third projection is located in the middle frequencies. Moreover, Panel 18b) shows a feature that is located at the higher frequencies of the spectrogram for all projections. A closer inspection of this feature and other features that show the same behaviour, reveals that the maximum activity of these features is negative for all average input phones. Thus, when placed on patterns of high activation, these features will have a strong negative activation. It could be that these features are sensitive to certain patterns of (close to) zero activation in the input phones.[11]

Again, both the sum of squared errors and the average column-wise cross correlation are computed between the features to see how similar they are. Table 2 shows the results for the second experiment. The results are the same as those of the first experiment. For both measures, the within-feature similarity of the first layer features is very high, whereas the between-feature similarity is lower. Furthermore, for the features of the second and third layer, the within-feature similarity is only a bit higher than the between-feature similarity.

Now that, in the two experiments, two different methods are used to visualize the same features, one could wonder to what extend these two methods give similar projections per feature. Table 3 shows this. As can be seen, the projections of the features of the first layer do not differ between the experiments. The projections of the features of the second and third layer do. However, the similarities of the between-method comparison are for both experiments closer to the within-feature similarities than to the between-feature similarities. Thus, it is likely that the difference between the projections of both experiments is caused by the variation in input phones used for deconvolution that also occurs in both experiments separately.

Table 2

Per layer, the average between-feature and within-feature similarity scores for the projections created in experiment 2. Similarity is measured as the sum of squared errors (SSE) and as the average column-wise cross correlation (corr).

	Between-feature		Within-feature	
Layer	SSE	corr	SSE	corr
1	14.3590	0.4355	<0.0001	1.0000
2	32.4422	0.5251	29.5021	0.5782
3	48.3679	0.5029	47.6350	0.5127

Table 3

Per layer, the average between-method similarity of all projections, which compares the projections created in experiment 1 with those created in experiment 2. Similarity is measured as the sum of squared errors (SSE) and as the average column-wise cross correlation (corr).

	Between-method	
Layer	SSE	corr
1	<0.0001	1.0000
2	29.8113	0.5928
3	44.6686	0.5223

[10] Again, all other projections are send along with this thesis.

[11] Note that in the first experiment, only the features that had a positive maximum activation were selected. Features that did not meet this requirement were not visualized.

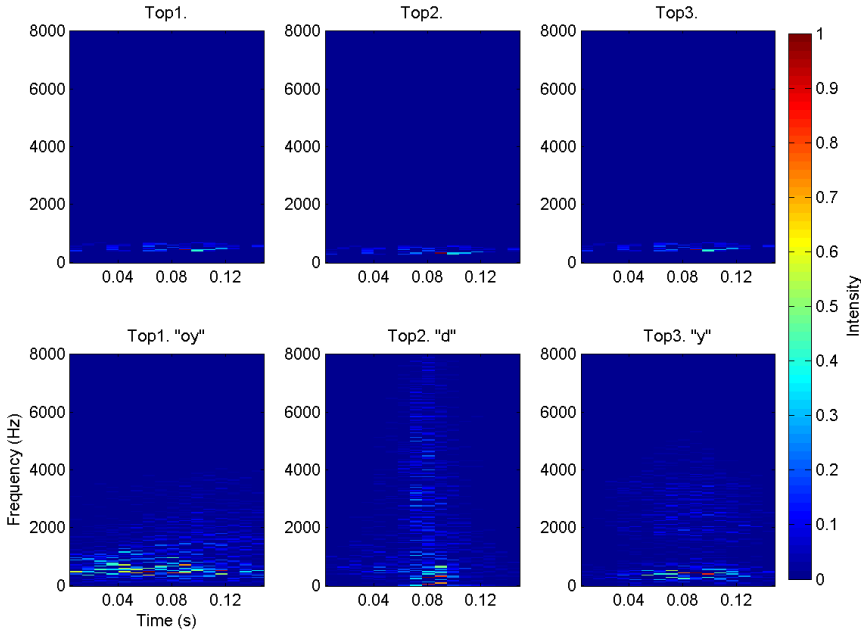
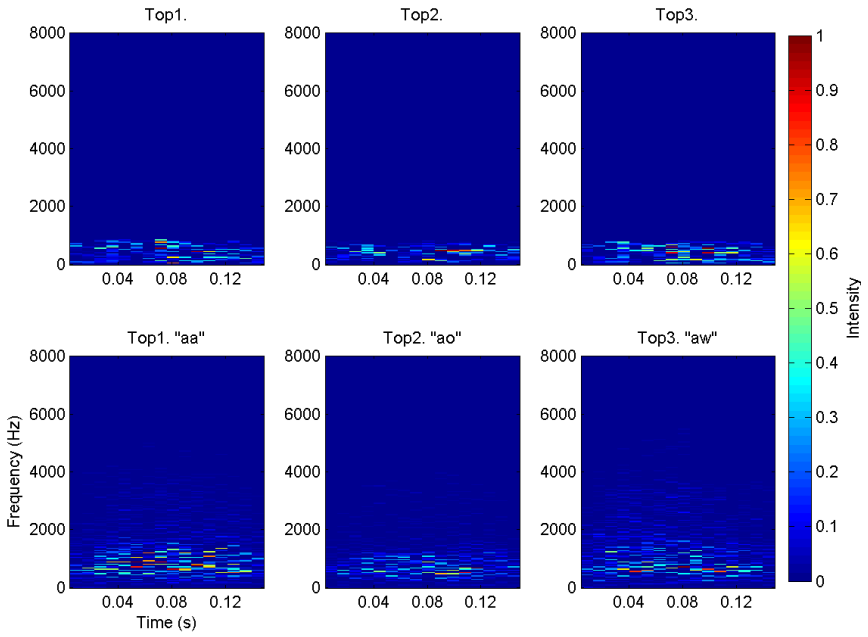
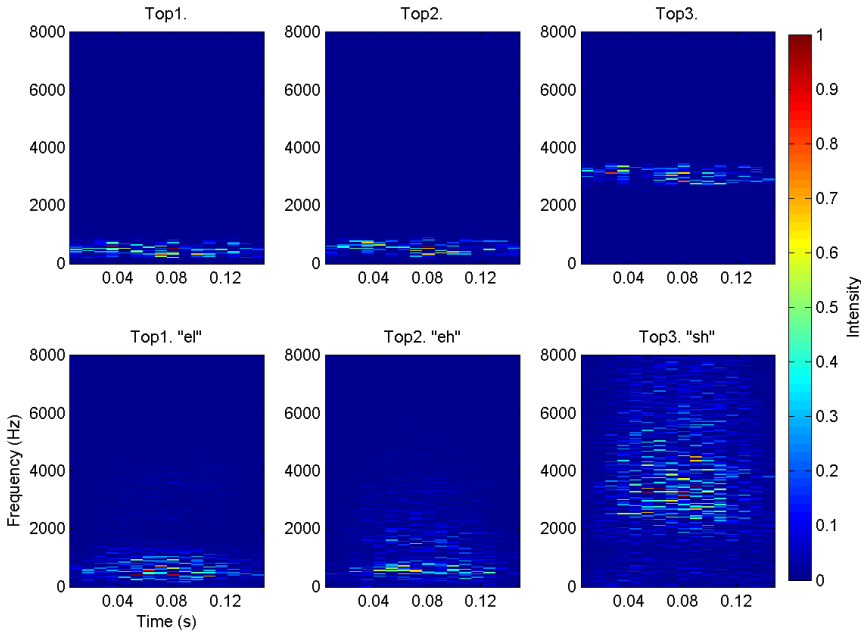


Figure 16. The projections and input phones of one out of eight features of the first convolutional layer.

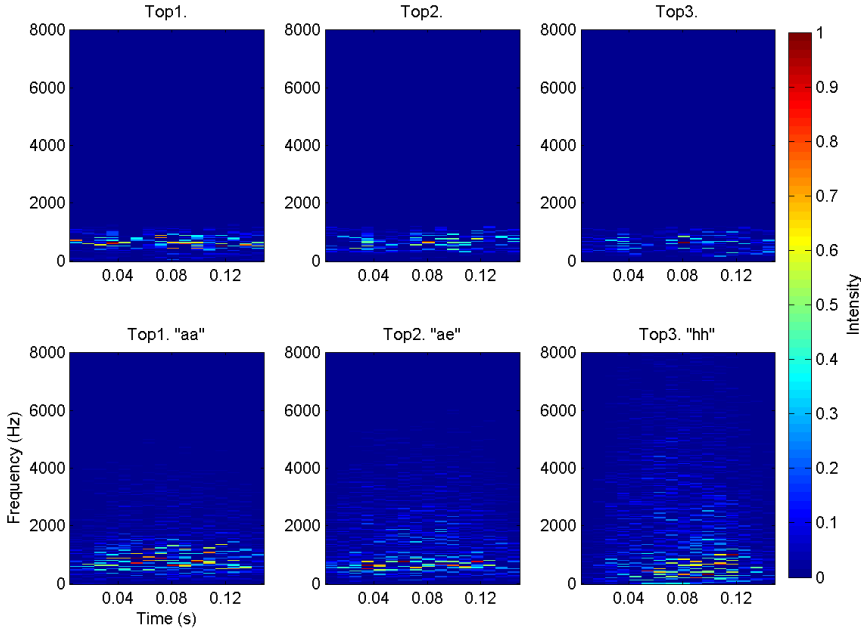


(a)

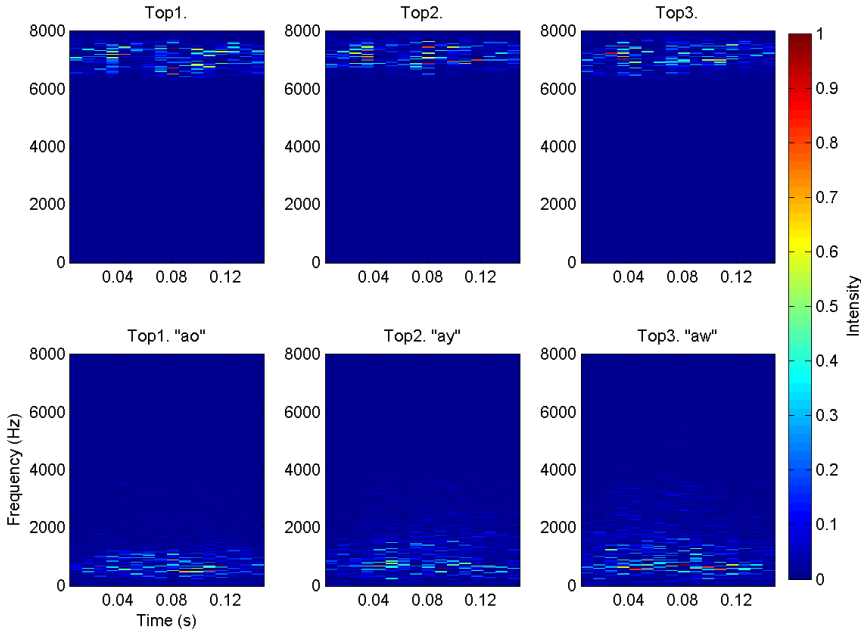


(b)

Figure 17. The projections and input phones of two out of sixteen features of the second convolutional layer.



(a)



(b)

Figure 18. The projections and input phones of two out of 32 features of the third convolutional layer.

Phone label analysis. Figure 19 shows how often each phone label is used as an input phone for the deconvolution process in the second experiment. As can be seen, besides vowels also multiple consonants appear often in the top 3 activations of features. This was not the case in the first experiment. Probably, the reason for this change is that the intensity of all input spectrograms were set to vary in the same range.

Thus, the advantage of the second experiment over the first experiment is that the phone labels of the input phones can give information about the sensitivity of features to certain types of phones. Table 4 shows per layer how many features are mainly sensitive to consonants, to vowels or to neither. The sensitivity of a feature is based on the 5 average input phones that cause the feature to be most activated. A feature is said to be sensitive to consonants, if at least 3 out of 5 input phones is a consonant, and at most 1 out of 5 input phones is a vowel[12]. A feature is said to be sensitive to vowels, if at least 3 out of 5 input phones is a vowel, and at most 1 out of 5 input phones is a consonant. As can be seen in the table, the features of the first layer are about equally sensitive to consonants and vowels. The features of the second layer are mainly sensitive to neither consonants or vowels and some are sensitive to vowels; only one feature is sensitive to consonants. The features of the third layer are mainly sensitive to vowels, and equally sensitive to neither or to consonants.

And if a feature is sensitive to, say consonants, is this a specific type of consonant it is sensitive to? For example, consonants with the constriction located in the front of the vocal tract compared to those with the constriction located in the back of the vocal tract? Or, in case of vowels, are there features sensitive to front or back vowels in specific? This has been analysed for the features of the third convolutional layer. Each vowel and each consonant has been graded on a 7 point scale, with 1 being very front and 7 being very back [13]. For each vowel-sensitive feature, the scores of its top 3 input vowels are added up to a total score. For each consonant-sensitive feature, the same is done for the scores of its top 3 input consonants. Thus, the feature can have a total score ranging from 3 to 21, with 3 meaning that it is very sensitive to front vowels or consonants and 21 meaning that it is very sensitive to back vowels or consonants. Scores of around 12 mean that the feature is not in specific sensitive to either front or back vowels or consonants.

Table 4
Per layer, the number of features that is mainly sensitive to consonants or to vowels or to neither.

Layer	consonants	vowels	neither	total
1	3	4	1	8
2	1	5	10	16
3	7	18	7	32

Figure 20 shows the results of this analysis. Panel 20a) shows the results for the 18 vowel-sensitive features of the third convolutional layer. It can be seen that some features are mainly sensitive to front vowels and others are mainly sensitive to back vowels. Panel 20b) shows the results for the seven consonant-sensitive features. Five out of seven features do not seem to be sensitive to either front or back consonants and two out of seven features seem to be more sensitive to back consontants.

[12] There are input phone labels that are neither a consonant nor a vowel, such as 'sil' which refers to 'silence'.

[13] This grading was based on second chapter of the book by Harley (2008). All front vowels scored 1, all central vowels scored 4 and all back vowels scored 7 points. All bilabial, labiodental and dental consonants scored 1, all alveolar consonants scored 4 and all postalveolar, velar and glottal consonants scored 7.

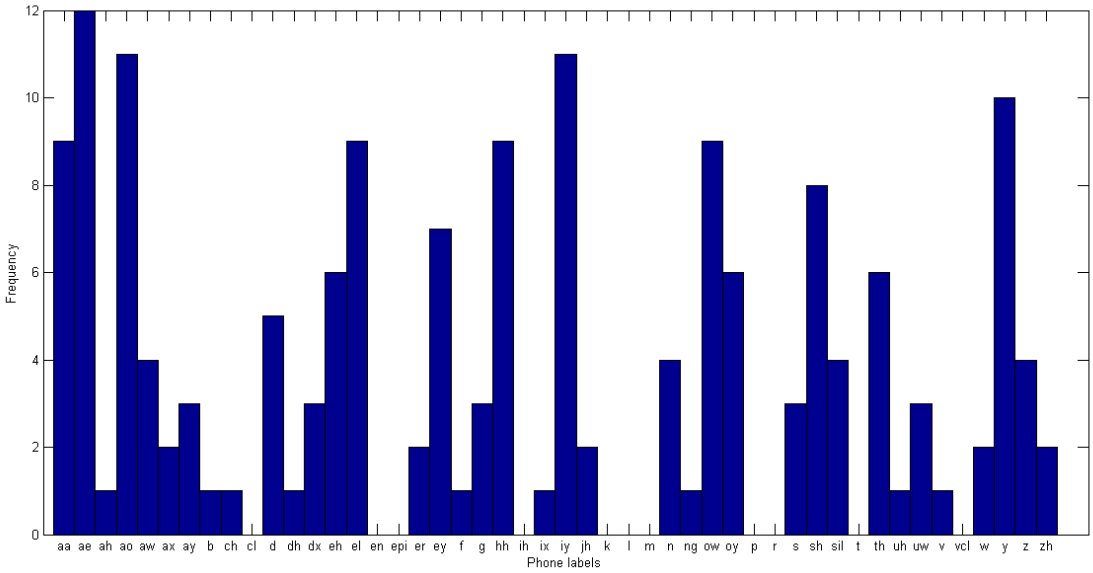
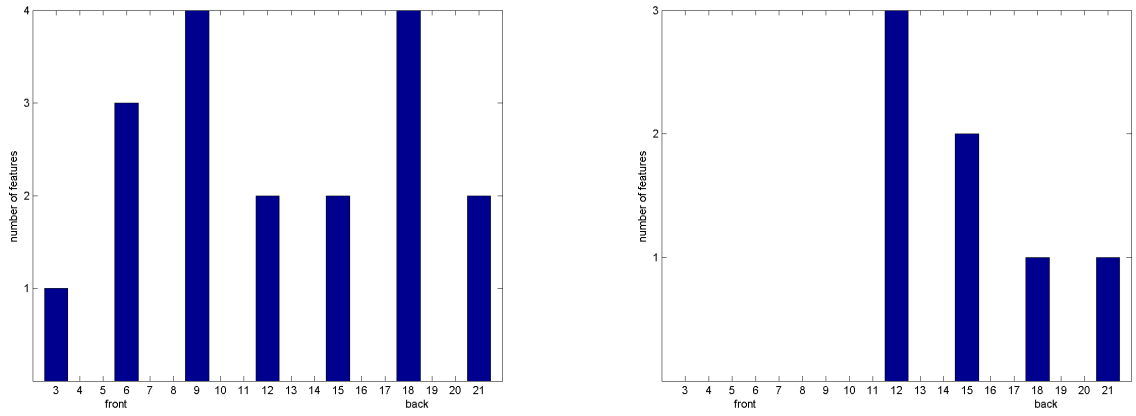


Figure 19. Histogram that plots how often each phone label appeared in the top 3 average phones of features (experiment 2).



(a) (b)

Figure 20. These histograms show how many features of the third layer are sensitive to specific types of vowels and consonants. Panel a) shows for all vowel-sensitive features, whether they are specifically sensitive to front or to back vowels. Panel b) shows for all consonant-sensitive features, whether they are specifically sensitive to consonants with the constriction located in the front or in the back of the vocal tract.

Discussion

This study searched for an answer on the following two research questions. First, what do the features of a convnet represent when the convnet is trained to classify phones? And second, how do the features change from layer to layer?

Let us first tackle the second research question, because it is the easiest to answer. The hypothesis was that features will get more complex when moving from the first to the latter layers, and the results seem to confirm this hypothesis: The features of the second layer look more complex than the features of the first layer. For example, one feature of the second convolutional layer (feature (d) in Figure 8) is sensitive to activity both in the onset and end of a phone, whereas all features of the first layer are sensitive to activity at only one position in time. Also the features of the third layer look more complex than the features of the second layer. For example, some features of the third layer (amongst others, feature (c) in Figure 9) seem to be sensitive to two horizontal bands of activation.

But of course, it is not strange that the features get more complex from layer to layer. First of all, the features of the latter layers cover more frequency bands of the input and thus have more 'space' to show complex patterns. Second, the whole point of a convnet is that latter features combine earlier features into a more complex representation. If this hypothesis would not have been confirmed, there would probably be an error in the implementation of the convnet.

Let us now deal with the more difficult question: what do the features of the convnet represent? The hypothesis was that the features in the first layer represent formants, their onset and end points and random noise patterns, and that the features in the latter layers represent the position of noise and formants relative to each other and their frequencies. When looking at the projections of the features, formant-like patterns can be found. Also, there seem to be some features in the latter layers that show their relative positions. However, these patterns never show clearly in all projections of the feature. As the similarity measures indeed show, the projections per feature are only slightly more similar than the projections of different features. It is thus difficult to draw conclusions about the meaning of the features based on their projections.

Experiment 2 was started for this reason. The idea was that if the input phones would really match the features, then it would be easier to interpret the feature projections themselves. In the study of Zeiler & Fergus (2014) this was the case. They depicted projections of features that are much more easy to interpret when looking at the input image that was used for the projection. In the current study, the features are slightly more easy to interpret when also looking at the input phones that were used for the projections. For example, for some features the input phones show that they are mainly sensitive for consonants or for shorter phones. However, the benefit of seeing the input phones is not as high as in the study of Zeiler & Fergus (2014). One reason for this is that visually depicted sound is not as easy to interpret as depicted images. Moreover, in the study of Zeiler & Fergus (2014) there was much more variation in input images and thus it is informative to see to what part of the image a feature was most sensitive. Whereas in the current study, most features are sensitive to the same lower frequencies of the input phone.

Thus, this research question cannot be answered based on the results of the current study. How could this question be answered in future research? A first suggestion would be to apply deconvolution to a convnet that has more than 3 convolutional layers. Maybe latter layers will show more clear patterns. Unfortunately, in the current study the resources were not available for a larger network. A second suggestion would be to apply deconvolution to a convnet that uses convolution across the time-axis instead of across the frequency-axis. In the current study, convolution across the frequency-axis was used, which caused features to have formant-like shapes. This way, it is very difficult to discriminate between a "regu-

lar" feature and a feature that represents a formant. When using convolution across the time-axis this problem does not arise. Features will be vertical bars and formants would be recognizable by horizontal bars of sensitity across the feature. If there are no horizontal bars of sensitivity, then features simply do not represent formants.

An advantage of experiment 2 is that the labels of the phones used for deconvolution vary from feature to feature and thus can be used for deducing the meaning of the features. Experiment 2 showed that some features were mostly activated by vowel inputs whereas other features were mostly activated by consonant inputs. This finding could help explain the results found by Ma et al. (2014). As described in the introduction, they trained a deep neural network on phone recognition and found that the first layers were more responsible for recognizing back vowels and front consonants, whereas the latter layers were more responsible for recognizing front vowels and back consonants. In the current study, it was found for the third convolutional layer that there were indeed features that were more sensitive to one of both types of vowels. Moreover, there was a tendency for consonant-sensitive features to be more sensitive to back consonants than to front consonants. However, there were not enough consonant-sensitive features to see clear patterns. Furthermore, in the study by Ma et al. (2014) a deep neural network with seven layers was used, whereas in the current study, only the three convolutional layers could be analysed, of which only the third layer had enough features to draw some conclusions. Thus, the results of this study cannot be used for understanding why the responsibility of the features differ from layer to layer as found by Ma et al. (2014). Future research could delve deeper into this, by applying phone label analysis to convnets with more convolutional layers and more features.

To conclude, the current study has explored the deconvolution method for visualizing the features of a convnet trained for phone recognition. Visualization was easy, but interpretation was hard. It could be that features of convnets with other architectures are more easy to understand visually. Moreover, other methods could be considered in order to understand what the features represent. For instance, one could explore the types of phones that each feature is sensitive to. Despite all questions that still remain, this study has contributed to the small field of research on the representations of speech in convnets and hopefully it has taken us one small step closer to the answers.

References

Abdel-Hamid, O., Deng, L., & Yu, D. (2013). Exploring convolutional neural network structures and optimization techniques for speech recognition. *INTERSPEECH*, 3366-3370.

Abdel-Hamid, O., Mohamed, A., Jiang, H., & Penn, G. (2012). Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. *Proc. ICASSP*, 4277-4280.

Churchman, T. J. (2015). *Reconstructing speech input from convolutional neural network activity* (Unpublished bachelor thesis). Radboud University Nijmegen, The Netherlands.

Deng, L., Hinton, G., & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Dyer, C. (n.d.). *Notes on adagrad.* Retrieved 2015-05-27, from `www.ark.cs.cmu.edu/cdyer/adagrad.pdf`

Garofolo, e. a., John. (1993). *TIMIT acoustic-phonetic continuous speech corpus LDC93S1.* Web Download. Philadelphia: Linguistic Data Consortium.

Gibiansky, A. (2014a). *Convolutional neural networks.* Retrieved 2015-03-13, from `http://andrew.gibiansky.com/`

blog/machine-learning/convolutional-neural-networks/

Gibiansky, A. (2014b). *Fully connected neural algorithms.* Retrieved 2015-03-13, from http://andrew.gibiansky.com/blog/machine-learning/fully-connected-neural-networks/

Harley, T. A. (2008). *The psychology of language: From data to theory* (3rd ed.). Hove, East Sussex, UK: Erlbaum (UK): Taylor & Francis.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., ... Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, *29*(6), 82-97.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems (NIPS), Lake Tahoe, Nevada*.

Ladefoged, P., & Johnson, K. (2011). *A course in phonetics* (6th ed.). Cengage.

LeCun, Y., & Bengio, Y. (1995). The handbook of brain theory and neural networks. In M. A. Arbib (Ed.), (chap. Convolutional networks for images, speech, and time-series). MIT Press.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 2278-2324.

Lee, K. F., & Hon, H. W. (1988). Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Audio, Speech and Language Processing*, *37*(11), 1641-1648.

Lopes, C., & Perdigao, F. (2011). *Speech technologies* (P. I. Ipsic, Ed.). InTech. (ISBN: 978-953-307-996-7, Available from: http://www.intechopen.com/books/speech-technologies/phoneme-recognition-on-the-timit-database)

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

Matlab audio database toolbox. (n.d.). Retrieved 2015-03-21, from http://nl.mathworks.com/matlabcentral/fileexchange/23843-matlab-audio-database-toolbox

Ng, A., Ngiam, J., Foo, C. Y., Mai, Y., Suen, C., Coates, A., ... Tandon, S. (n.d.). *Ufldl tutorial.* Retrieved 2015-03-13, from http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

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

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for simplicity: the all convolutional net. *arXiv:1412.6806v3*.

Vedaldi, A., & Lenc, K. (2014). Matconvnet – convolutional neural networks for matlab. *CoRR*, *abs/1412.4564*. (version from February, 2015)

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions On Acoustics, Speech, and signal processing*, *37*(3), 328-339.

Zeiler, M., & Fergus, R. (2014). Visualizing and understanding convolutional networks. *arXiv:1311.2901*.

Appendix

Mathematical background of deep neural networks and convolutional neural networks

In this section, the mathematical background of deep neural networks and convolutional neural networks is described. This section is mainly based on Andrew Gibiansky's work (Gibiansky, 2014b,a). Moreover, the part about backpropagation for subsampling layers is also based on the UFLDL tutorial of Stanford University (Ng et al., n.d.). First, the mathematics behind ordinary deep neural networks is described. Then, the math behind convnets is described.

The following mathematical notations will be used:

- x_i^l refers to the total input of unit i in layer l.
- y_i^l refers to the output of unit i in layer l.
- w_{ij}^l refers to the weight from unit i in layer l to unit j in layer $l+1$.
- As can be seen in the notations, the layers are numbered. If we exclude the input layer, the total number of layers is equal to L. Thus L refers to the output layer.

Deep neural networks. A Deep neural network is a network that has multiple layers of units. All the units of a layer are connected to all units of the subsequent layer. Each of these connections has its own weight. The input layer is a 'special' layer. It is there to represent the input of the network. The units in this layer do not have any input, they only give output to the subsequent layer. Namely, their output is the input of the network. For example, if the input of the network is an image, the output of an individual input unit is the gray-value of a pixel. For all 'normal' subsequent layers, the input and output of unit i in layer l are computed as follows:

$$x_i^{(l)} = \sum_j w_{ji}^{(l-1)} y_j^{(l-1)}$$

$$y_i^{(l)} = f(x_i^{(l)})$$

The input of a unit is thus computed by summing over all units of the previous layer. Function $f()$ is the activation function of the layer. This function is the same across all units of the layer. In the current study the rectified linear function is used as the activation function for all layers of the network.

The output of the last layer is the output of the whole network. During training, the network is given multiple example inputs for which the correct output is known. Every time the network has runned through all example inputs, the weights of the network are changed in such a way that the error of the network decreases. The error of the network is based on the difference between the correct output and the output that the network has given. There are multiple possible definitions for this error, but for now those are not important. From now on, E refers to the error function.

To make the error of the network decrease, the weights are trained using the gradient descent algorithm. This algorithm makes use of a method called backpropagation. Backpropagation happens in six steps:

1. The forward pass. Compute the input and output of all layers for all training examples.
2. For all output units i, compute the derivates of the error with respect to the output y_i of that unit:

$$\frac{\delta E}{\delta y_i^{(L)}} = \frac{\delta E(y^{(L)})}{\delta y_i^{(L)}}$$

3. Compute the deltas of the current layer. The deltas are the partial derivates of the error with respect to the inputs of the layer. These deltas will later be used for computing the derivatives with respect to the weights of this layer. , The deltas can be computed using the following formula.

$$\frac{\delta E}{\delta x_i^{(l)}} = \frac{\delta E}{\delta y_i^{(l)}} \frac{\delta y_i^{(l)}}{\delta x_i^{(l)}} = \frac{\delta E}{\delta y_i^{(l)}} \frac{\delta f(x_i^{(l)})}{\delta x_i^{(l)}} = \frac{\delta E}{\delta y_i^{(l)}} f'(x_i^{(l)})$$

4. In order to compute the deltas for the previous layers, the error of the current layer must be backpropagated to the previous one. This can be done by using the following formula. Note that the formula sums over the inputs of all units j of layer l because all units in layer l use the output of unit i in layer $l-1$.

$$\frac{\delta E}{\delta y_i^{(l-1)}} = \sum_j \frac{\delta E}{\delta x_j^{(l)}} \frac{\delta x_j^{(l)}}{\delta y_i^{(l-1)}}$$

$$= \sum_j \frac{\delta E}{\delta x_j^{(l)}} \frac{\delta w_{ij}^{(l-1)} y_i^{(l-1)}}{\delta y_i^{(l-1)}} = \sum_j \frac{\delta E}{\delta x_j^{(l)}} w_{ij}^{(l-1)}$$

5. Repeat steps 3 and 4 to compute all errors and deltas for all layers besides the input layer.

6. Compute the derivatives of the error with respect to the weights for each layer, by using the deltas that were computed in step 3. These derivatives will later be used to update the weights.

$$\frac{\delta E}{\delta w_{ij}^{(l)}} = \frac{\delta E}{\delta x_j^{(l+1)}} \frac{\delta x_j^{(l+1)}}{\delta w_{ij}^{(l)}}$$

$$= \frac{\delta E}{\delta x_j^{(l+1)}} \frac{\delta w_{ij}^{(l)} y_i^{(l)}}{\delta w_{ij}^{(l)}} = \frac{\delta E}{\delta x_j^{(l+1)}} y_i^{(l)}$$

The results of the backpropagation algorithm are used in the gradient descent algorithm. This algorithm is as follows:

1. Set $\Delta W^{(l)} := 0$ for all layers l.

2. For all m training examples and for all layers l do:

   - Use backpropagation to compute the gradients $\nabla W^{(l)}$. Note that $\nabla W^{(l)}$ is the matrix of all gradients of layer l.
   - Set $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}}$.

3. Update the parameters by using the following formula, in which α is the learning rate and m is the number of training examples.

$$W^{(l)} = W^{(l)} - \alpha(\frac{1}{m} \Delta W^{(l)})$$

Convolutional neural networks. Convolutional neural networks have two types of distinctive layers: convolutional layers and sub-sampling layers. First, the math behind both kinds of layers is described. Then, it is described how backpropagation works for these layers.

Suppose that we have a convolutional neural network of which a part is shown in Figure 21. Its input layer has size $N \times N$ and its first convolutional layer has four feature maps, each with its own weight matrix, which are called filters. All filters have size $m \times m$. The input for a unit in one of the feature maps within this convolutional layer is given by the following formula.

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{ab} y_{(i+a)(j+b)}^{(l-1)}$$

Note that w_{ab} refers to the weight on location (a, b) in filter w, the filter that belongs to the feature map of this unit. As can be seen in Figure 21, the size of the filter w determines the size of the receptive field of the unit. Thus, what this formula does is computing the input of the unit within the convolutional layer by going through all the units within its receptive field, multiply their outputs by the corresponding weights of filter w and summing over all these results.

The output y_{ij}^l of the unit within the convolutional layer is computed by applying the activation function to the input of this unit:

$$y_{ij}^l = f(x_{ij}^l)$$

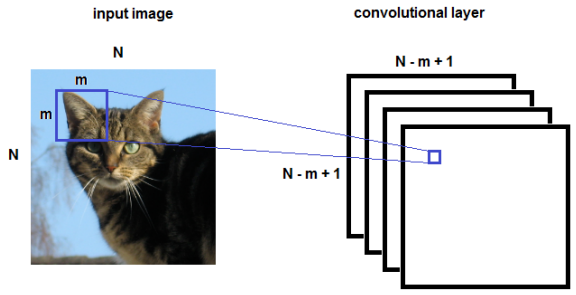


Figure 21. The receptive field of a unit within a convolutional layer.

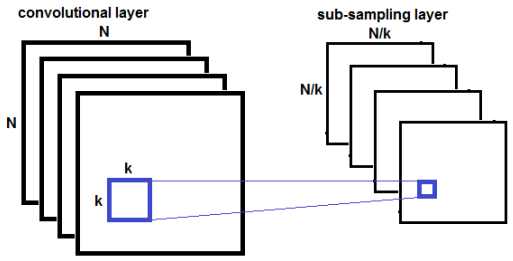


Figure 22. The receptive field of a unit within a sub-sampling layer.

Compared to the convolutional layer, the mathemetical methods of the sub-sampling layer is quite easy. As can be seen in Figure 22, each feature map in the convolutional layer has its sub-sampled version in the sub-sampling layer. Each unit in the sub-sampling layer has a receptive field within the corresponding feature map in the convolutional layer. The output of the sub-sampling unit is either the average (in case of average pooling) or the maximum (in case of max pooling) of the outputs of all units within its receptive field. Often, the receptive fields of the sub-sampling units within one feature map do not overlap. Suppose the size of the feature maps is $N \times N$ and the size of the receptive fields of the sub-sampling units is $k \times k$. Then, given that the receptive fields do not overlap, the feature maps within a sub-sampling layer have a size of $N/k \times N/k$.

Also for the training of a convolutional neural network the backpropagation algorithm is used to compute the needed gradients. For all fully connected layers within the convolutional neural network, backpropagation works as was explained in the previous section. For the convolutional and sub-sampling layers the algorithm is slightly different.

Backpropagation for sub-sampling layers is very easy, because sub-sampling layers do not have any weights that need to be learned. Therefore, the error should only be backpropagated to the layer in front of the sub-sampling layer. In case of max pooling, only the unit that had the highest output value of all units within the receptive field of the sub-sampling unit influences the output of the sub-sampling unit. Thus, all error is backpropagated to this unit. In case of average pooling, all units within the receptive field of the sub-sampling units have equal influence on the output of the sub-sampling layer. And thus, every unit within the receptive field gets an equal share of the backpropagated error.

Backpropagation for convolutional layers is more difficult. As can be seen in the previous description of backpropagation we need three partial derivatives of the error in order to both train the weights of the convolutional layer and backpropagate the error to the previous layer. These three partial derivatives are:

1. The deltas, which are the partial derivatives of the error with respect to the input of the layer:

$$\frac{\delta E}{\delta x_{ij}^l} = \frac{\delta E}{\delta y_{ij}^l}\frac{\delta y_{ij}^l}{\delta x_{ij}^l} = \frac{\delta E}{\delta y_{ij}^l}\frac{\delta f(x_{ij}^l)}{\delta x_{ij}^l} = \frac{\delta E}{\delta y_{ij}^l} f'(x_{ij}^l)$$

2. The partial derivatives of the error with respect to the output of the previous layer:

$$\frac{\delta E}{\delta y_{ij}^{l-1}} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\delta E}{\delta x_{(i-a)(j-b)}^l}\frac{\delta x_{(i-a)(j-b)}^l}{\delta y_{ij}^{l-1}}$$

$$= \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\delta E}{\delta x_{(i-a)(j-b)}^l}\frac{\delta w_{ab} y_{ij}^{l-1}}{\delta y_{ij}^{l-1}}$$

$$= \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\delta E}{\delta x^l_{(i-a)(j-b)}} w_{ab}$$

Note that, just like with normal backpropagation, the formula sums over the inputs of all units in layer l that use the output of unit (i, j) in layer $l-1$. Figure 23 explains where the coordinates $(i-a)$ and $(j-b)$ come from. Suppose the red unit in the input layer has output y^{l-1}_{ij}. This output is used by the four orange units in the convolutional layer, given that the size of the receptive field is $m \times m = 2 \times 2$. For example, for the upper left orange unit, the red unit is in the lower right corner of its receptive field. Since the output of the red unit is used by the four orange units, the gradient of the error with respect to the output of the red unit should be computed by back propagating the error through the four orange units. Note that the coordinates of the lower right orange unit are (i, j) and therefore the coordinates of all orange units can be described as $(i-a, j-b)$ with a and b ranging from 0 to $m-1$.

Also, note that this formula does not work for the derivatives of the error with respect to the output of the units at the left or upper edge. Take for example the upper left unit in the input layer in Figure 23. The output of this unit only has an influence on the input of the upper left unit in the convolutional layer. This problem is solved by padding the left and upper edges of the convolutional layer with zeros. This way, the formula also works for units at the left or upper edge.

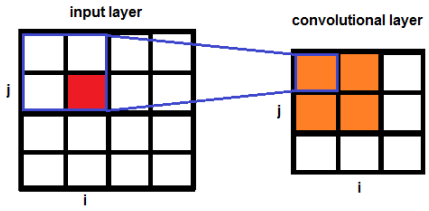


Figure 23. The output of the red unit in the input layer is used by the four orange units in the convolutional layer, given that the size of the receptive fields of the convolutional units is 2×2.

3. The derivatives of the error with respect to the weights:

$$\frac{\delta E}{\delta w_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\delta E}{\delta x^l_{ij}} \frac{\delta x^l_{ij}}{\delta w_{ab}}$$

$$= \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\delta E}{\delta x^l_{ij}} \frac{\delta w_{ab} y^{(l-1)}_{(i+a)(j+b)}}{\delta w_{ab}}$$

$$= \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\delta E}{\delta x^l_{ij}} y^{(l-1)}_{(i+a)(j+b)}$$

Note that the formula sums over the inputs of all units in layer l that use weight w_{ab} from layer $l-1$.

Besides these three partial derivates, a convolutional neural network can be trained in the same way as a deep neural network. Use these three partial derivates in the backpropagation algorithm as described in the previous section. Then, use the gradient descent algorithm to train the network.