Proceedings of Machine Learning Research 123:192–202, 2020 NeurIPS 2019 Competition and Demonstration Track

Visualizing and sonifying how an artificial ear hears music

Vincent Herrmann VINCENT.HERRMANN@WEB.DE
Institute for Music Informatics and Musicology,
University of Music Karlsruhe

Editors: Hugo Jair Escalante and Raia Hadsell

Abstract

A system is presented that visualizes and sonifies the inner workings of a sound processing neural network in real-time. The models that are employed have been trained on music datasets in a self-supervised way using contrastive predictive coding. An optimization procedure generates sounds that activate certain regions in the network. That way it can be rendered audible how music sounds to this artificial ear. In addition, the activations of the neurons at each point in time are visualized. For this, a force graph layout technique is used to create a vivid and dynamic representation of the neural network in action.
Keywords: Audio processing, visualization, self-supervised learning

1. Introduction

Although neural networks are often described as black boxes, there exist methods to make us see and hear what is going on inside a neural network. A well-known example of this is the DeepDream technique (Mordvintsev et al., 2015) that generates bizarre psychedelic but strangely familiar pictures. We follow a related approach to make audible the innards of sound processing neural nets.

The first thing we need is a model we want to examine. We train it on musical data using a self-supervised training algorithm, namely mutual information maximazation using contrastive predictive coding (Hjelm et al., 2018; Oord et al., 2018).

Then the basic idea is the following: We start with an arbitrary sound clip (e.g. a drum loop). This clip will be modified by an optimization procedure in a way that stimulates a certain region in the neural net. The changed clip can then be used as a basis for a new optimization, potentially with a different target region. This way you obtain sounds that are generated by the neural network in a freely associative way.

One of the most important properties of neural networks is that they process information on different, progressing levels of abstraction (LeCun et al., 2015). This has been shown for image processing networks using algorithmic methods (Kozma et al., 2018), as well as feature visualizations (Olah et al., 2017). If we try to transfer the concept of hierarchical abstractions to the musical domain, it is reasonable to expect some regions of the net being associated with, for example, short and simple sounds or noises, and other regions with more complex musical phenomena, such as rhythm or harmony. For what exactly a neural networks listens depends of course on the data it was trained on, as well as on the task it is specialized to accomplish. This work allows discovering both visually and auditory the

© 2020 V. Herrmann.

features a particular model relies on. For that, a is setup developed that allows — not unlike a DJ mixing console — the generation, visualization and control of sound in real-time.

The contributions presented here are the following:

- The application and evaluation of contrastive predictive coding (Oord et al., 2018) for self-supervised representation learning on spectral representations of music. For this, fully convolutional architectures for both encoder and autoregressive model are used.
- A method of finding a clear visual representation of the neural activity in a deep learning model using the multilevel force graph layout technique and visualizing activations of all neurons in real-time.
- Transfer of the principles of feature visualization (Erhan et al., 2009; Mordvintsev et al., 2015; Olah et al., 2017) to the sound domain, enabling the sonification of features a model relies on.
- Development of a system that allows the real-time control and exploration of the sonic feature space.

Audio and video examples, as well as further resources, are available online[1].

2. Model

2.1. Contrastive Predictive Coding

In the centre of this project stands an "artificial ear", meaning a sound processing neural network. It has been shown that analogies exist between convolutional neural networks (ConvNets) and the human auditory cortex (Kell et al., 2018). Of course, the human auditory system is not a classification network, like the one used in Kell et al. (2018), we usually do not learn with the help of explicit labels. The question of the actual learning mechanisms in the brain is highly contested, but a promising candidate, especially for perceptual learning, might be predictive coding (Rao and Ballard, 1999; Friston, 2005). Here future neural responses to stimuli are predicted and then compared to the actually occurred responses. The learning entails reducing the discrepancy between prediction and reality.

One method of using the idea of predictive coding for self-supervised learning in artificial neural networks is presented in Oord et al. (2018) and Ozair et al. (2019). For this so-called contrastive predictive coding, you need two models: The encoder model builds a compressed encoding $\mathbf{z}_t$ of the signal for each time step t. The autoregressive model summarizes multiple sequential encodings into a vector $\mathbf{c}$. The encoding k time steps in the future, $\mathbf{z}_k$, is predicted by a different learnt linear transformation $\mathbf{M}_k$ of $\mathbf{c}$ for each time step. The time steps k are counted from the latest point t for which the autoregressive model had access to the encoding $\mathbf{z}_t$. The loss for each mini-batch is calculated as follows:

$$\mathcal{L}_{CPC} = -\sum_{n,k} \log \frac{\exp(\mathbf{z}_{n,k}^T \mathbf{M}_k \mathbf{c}_n)}{\sum_m \exp(\mathbf{z}_{m,k}^T \mathbf{M}_k \mathbf{c}_n)}$$

1. https://vincentherrmann.github.io/blog/immersions/

Both n and m are indices of the mini-batch dimension. The numerator is a scalar score that measures how well the prediction matches the correct encoding. It is divided by the sum of the scores that measure the prediction against all encodings of the current mini-batch. Minimizing $\mathcal{L}_{CPC}$ maximizes the mutual information between $\mathbf{c}$ and future encodings $\mathbf{z}_k$. This means that there is little incentive to encode high-frequency noise since this information is not shared over longer time scales. Instead, the model can learn to focus on "slow" features that are useful for differentiating those signals.

Using contrastive predictive coding frees us to choose our training data without many constraints. No expensively labelled datasets are required, the model learns to make sense of any inputs it is given.

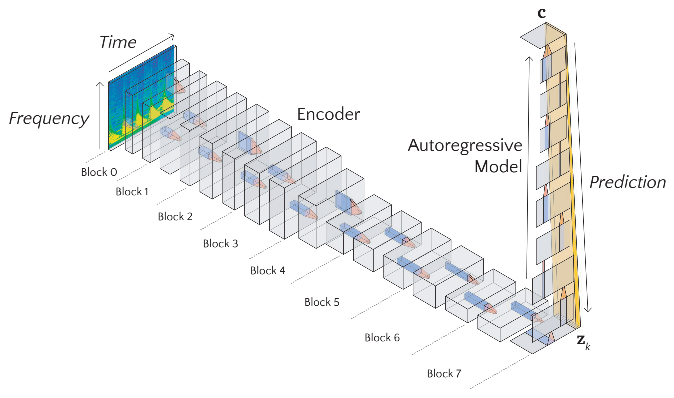

Figure 1: Encoder and autoregressive model architecture.

2.2. Architecture

The encoder network receives as input a scaleogram representation of an audio clip. A scaleogram is the result of a constant-Q or a wavelet transformation (Kjartansson, 1979) of the audio signal and comes quite close to the representation that the cochlea passes on to the brain (Kell et al., 2018). For the purpose of input optimization, which we will cover below, the transformation from a raw audio signal to the scaleogram has to be differentiable. Fortunately, it can simply be implemented as a 1D convolutional layer with fixed weights in any standard deep learning framework. For each frequency band, two channels of the kernels are assigned to represent the real and imaginary part of the corresponding finite impulse response filter. The stride parameter is equivalent to the hop length of the transformation. The output of this layer can be reshaped to have the dimensions time, frequency and real/imaginary. This representation is then converted into polar coordinates. The amplitude is squared and logarithmically scaled. This results in the two-dimensional scaleogram representation which is the input to the encoder network. We also use the phase part as the second channel for the input, although this did not noticeably improve the performance.

As encoder serves a 2D-ConvNet, similar to the kind common in computer vision and image processing. It is based on the ResNet architecture (He et al., 2016), with a few modifications: Since the model is trained on a sequence prediction task, we use only causal convolutions (Oord et al., 2016; Bai et al., 2018). In practice, this simply means that padding is done only at the start and not at the end of the time dimension for both convolution and pooling layers. Also, in addition to the usual quadratic kernels, we apply kernels that only span the frequency dimension and serve as overtone and harmony detectors. Details in Table 1(*a*).

At the output of the encoder, only the dimensions time and channel are left. They represent time step k and features of $\mathbf{z}_k$. In Oord et al. (2018), a recurrent neural network is used as the autoregressive model. We use a causal 1D-ConvNet instead, similarly to the encoder network employing ReLU nonlinearities, residual connections, max pooling and batch normalization — see Table 1(*b*).

Table 1: Number of channels, kernel sizes and pooling of the convolutions in the encoder and the autoregressive model.

(*a*) Encoder

Block	Channels	Conv 1	Conv 2
0	8	3x3 pool 2	1x25
1	16	1x3	1x3
2	32	3x3	1x15
3	64	1x3	1x3
4	128	3x3 pool 2	1x25
5	256	1x3	1x3
6	512	3x3	1x5
7	512	1x3	1x3

(*b*) Autoregressive Model

Block	Channels	Conv
0	512	4
1	512	4
2	512	1 pool 2
3	512	4
4	256	4
5	256	1 pool 2
6	256	4
7	256	1
8	256	4

2.3. Experiments

For the experiments, we use as input audio clips with a length of four seconds at a sample rate of 44.1 kHz. The scaleogram has a hop length of 1024 and 216 frequency bins (starting at 40 Hz with 24 bins per octave), resulting in the input size 172 x 216. The encoder model has two pooling layers with stride 2, which leads the time steps of the encodings $\mathbf{z}_t$ to have a length of ca. 93 ms. The autoregressive model integrates 42 consecutive encodings into one representation $\mathbf{c}$. From $\mathbf{c}$, 16 future encodings $\mathbf{z}_k$ are predicted, skipping the first 4 immediate next time steps because they pose no challenge.

The same model architecture was trained on two different datasets: one consisting of about 40 hours of house music mixes and the other one being the MAESTRO dataset (Hawthorne et al., 2018), containing 200 hours of classical piano music.

It is not necessarily evident just from a low value of $\mathcal{L}_{CPC}$ whether the model has indeed acquired any form of useful musical understanding. To evaluate this, the learned representations are used as features for two test tasks. Unfortunately, there are no suitable standard tests that are common in the raw audio and music domain, so the tasks used here should be seen more as a sanity check than as a way of completely and thoroughly assessing the learning algorithm's capabilities.

The first task is to assign audio clips to the house track it was extracted from (with a total of 15 possible tracks). For the second task the composer of the given audio clip has to be classified (the samples were taken from the MAESTRO validation set).

The test task audio data is fed into the trained models and one resulting feature **c** is saved for each short audio clip. Due to the receptive field of the encoder, one **c** vector represents ca. 1.8 seconds of sound, so there is no overlap between the items. These feature vectors are then provided with the corresponding labels (the track they came from or the composer) and partitioned into a training and an evaluation set. A linear classifier is fitted with the training set.

As a very simple baseline, the whole architecture was trained from scratch in a purely supervised setting for both of the tasks. For this, only the test task training data was used.

Also, the models were trained additionally with a slight modification of the CPC algorithm described in 2.1: Instead of contrasting the prediction scores with the other samples from the mini-batch, they are also contrasted with different time steps (score across time steps, SAT):

$$\mathcal{L}_{CPC\,SAT} = -\sum_{n,k} \log \frac{\exp(\mathbf{z}_{n,k}^T \mathbf{M}_k \mathbf{c}_n)}{\sum_{m,l} \exp(\mathbf{z}_{m,l}^T \mathbf{M}_l \mathbf{c}_n)}$$

Results can be found in Table 2. The performance of the CPC trained models on the test tasks is encouraging, especially when compared to the supervised models. Maybe not surprisingly, the latter show poor performance as there was not enough training data to learn meaningful features for the tasks. The use of SAT loss improves performance only for the model trained on the house dataset. One possible explanation might be that SAT training makes better use of limited data, but can be harmful if enough data is available. It could also depend on the specific nature of the data. To determine this, further investigation is needed.

Model	**house loss**	**house acc**	**maestro loss**	**maestro acc**
CPC house	0.3914	90.44%	2.212	25.36%
CPC house SAT	**0.3583**	**91.81%**	2.087	30.06%
CPC maestro	0.5131	86.83%	**0.659**	**77.49%**
CPC maestro SAT	0.8789	76.62%	0.700	75.90%
supervised house	1.563	48.25%		
supervised maestro			2.744	19.70%

Table 2: Results of the test tasks. Evaluation loss and accuracy are given for the house track classification and the maestro composer classification using the representation **c** as features for a linear classifier.

3. Visualization

To illustrate what is going on inside the model at each moment, we visualize the activation of the artificial neurons (i.e. the outputs of the nonlinearities). The place of each neuron in our network can be described by four coordinates: layer, frequency, time and channel. We will treat the time dimension different from the other ones and exclude it for now. The neurons can be seen as the vertices of a graph. They are connected in different patterns by edges, depending on the type of layer (e.g. fully connected or convolutional) the neurons belong to. There exist methods to lay out graphs into suitable shapes. For our purposes, the force layout technique (Walshaw, 2006; Hu, 2005) is the best fit. Here, the graph is modelled as a physical system with different forces acting on the vertices. The system, starting from random initial conditions, can be simulated using numerical methods, e.g. Verlet integration (Verlet, 1967).

In our setting, there are three types of forces acting on each vertex. The first force pulls together vertices that are directly connected. The strength is proportional to the distance between vertices and can be scaled with the weight of the connecting edge. Formally, the attractive force $\mathbf{a}$, caused by vertex j and acting on vertex i, with their respective positions $\mathbf{p}$, and connected by an edge with weight w_{ij}, is:

$$\mathbf{a}_{ij} = w_{ij}\,(\mathbf{p}_j - \mathbf{p}_i)$$

If the vertices i and j are not connected, w_{ij} is 0. In practice, the attractive forces are calculated only for connected vertices.

The second force works as if each vertex had a certain electric charge which would make vertices repel each other. The force $\mathbf{r}$ caused by vertex j and acting on vertex i, with their respective charges q, is:

$$\mathbf{r}_{ij} = q_i q_j \, \frac{\mathbf{p}_i - \mathbf{p}_j}{\|\mathbf{p}_i - \mathbf{p}_j\|^{k+1}}$$

For a physically accurate simulation of an electrostatic force, the exponent k would have to be 2. But seen as an adjustable parameter, it gives us some control over the appearance of the layout: It controls which of the forces, repulsive and attractive, dominates at a given distance between vertices. For high values of k, the repulsive force is strong over short and weak over long distances. This leads to the vertices being more evenly spread out. On the other hand, low values of k result in dense clusters of vertices (see Figure 2).

The optional centring force, scaled by a global factor l, pushes vertices towards the origin:

$$\mathbf{c}_i = -l\,\mathbf{p}_i$$

With that, the combined $\mathbf{f}$ force acting on vertex i, can be defined as:

$$\mathbf{f}_i = \mathbf{c}_i + \sum_{j \neq i} \mathbf{r}_{ij} + \mathbf{a}_{ij}$$

Since the number of individual forces that have to be calculated scales quadratically with the number of vertices, for large networks, it is necessary to approximate the forces

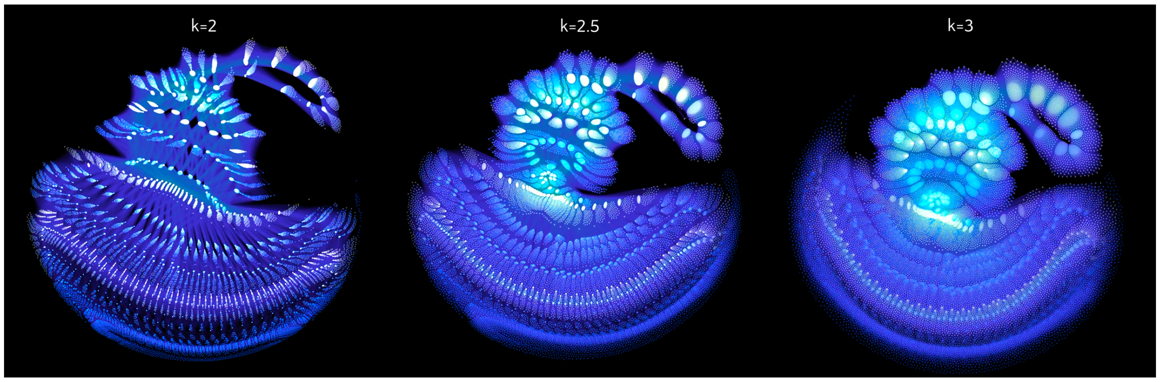

Figure 2: Layout of the network with three different attraction exponent parameters k.

$\mathbf{r}_{ij}$ in a computationally efficient way using the Barnes-Hut-tree method (Barnes and Hut, 1986).

To avoid local energy minima, it is helpful to start the simulation with a low-resolution graph and increasing the resolution in steps until the full graph reached. For our neural networks, the construction of lower resolution levels can be done by iteratively consolidating vertices that are neighbours in the channel or the frequency dimension. The expansion is done in the inverse order. This can be seen in Figure 3. Also, it can be helpful to slowly increase the centring force factor with each new resolution level.

There are many possibilities on how to determine the weights of the edges and the charges of the vertices. In this work, uniform weights are used for the edges and the charge of a vertex is set to be the variance of its corresponding neuron's activation across the validation set. For each point in the time dimension, which we excluded above, we get different charge values for the vertices, which results also in a slightly different layout. Calculating several of these snapshot layouts allows us to construct an animated version of the network. Once the layout is worked out the current state of the net can be depicted by lighting up strongly activated neurons and letting others stay dark. The code for the GPU accelerated layout calculation (implemented in PyTorch) and the OpenGL-based visualization is publicly available[2].

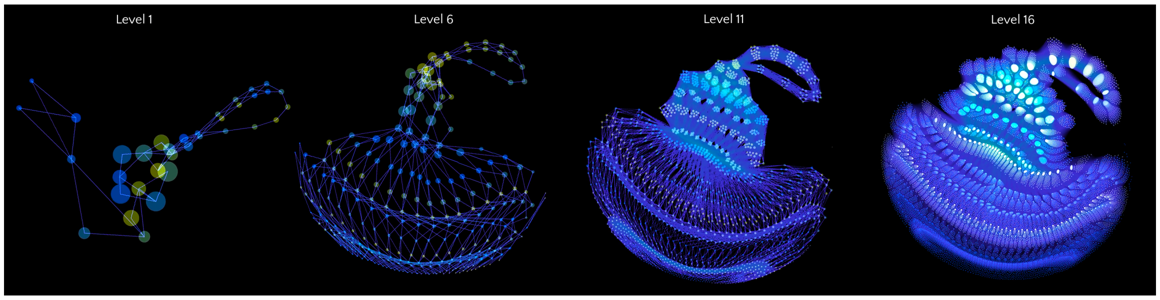

Figure 3: Force layout simulation at four progressive levels of resolution.

2. https://github.com/vincentherrmann/pytorch-graph-visualization

4. Input Optimization

Today's neural networks usually are completely differentiable. This means we can generate inputs for a trained model that maximize the activations of certain neurons in the network using iterative gradient-based optimization (Erhan et al., 2009; Mordvintsev et al., 2015; Olah et al., 2017). These inputs can then be directly experienced by humans and show which particular stimuli the selected neurons respond to. In our case, this procedure sonifies the features that activate a selected region in the network.

Neural networks, especially if they did not receive adversarial training Engstrom et al. (2019), are susceptible to small changes in the input. These can lead to local optima that are not perceptible by humans but still have the required properties. To prevent this we apply several types of regularization: temporal shifting of the input, small pitch changes, masking of random regions in the scaleogram, noisy gradients and de-noising of the input. All these methods make the input optimization more difficult in certain ways and thus enforce more robust and distinct results. In practice, there is no clear way to evaluate the perceptual quality of the generated inputs. Hence, finding the specific settings of the optimization procedure that work best for a certain model requires a certain amount of trial and error.

From figures 4 and 5, it can be seen that focus on the lower layers of the network does indeed result in more fine-grained and localized inputs. On the higher layers, however, it produces more organized rhythmic and harmonic patterns.

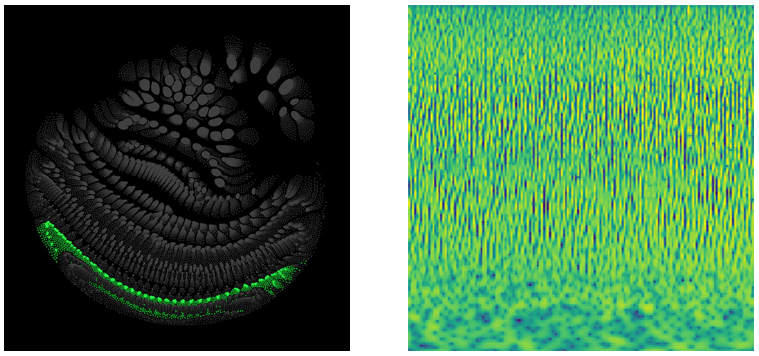
(a) *(b)*

Figure 4: Input optimization for the first convolution of encoder block 1. Selected neurons are highlighted in *(a)* and the scaleogram of the result is shown in *(b)*.

5. Live Performance

Like the inputs, the generated audio clips have a duration of about four seconds. With that, they lend themselves for a loop-based live performance. One loop then corresponds for example to two 4/4 measures at 120 bpm. The optimization procedure described above constantly generates new audio clips. As soon as one clip has finished playing, the latest newly calculated clip is started. An acoustic morphing arises as a result that is also reflected

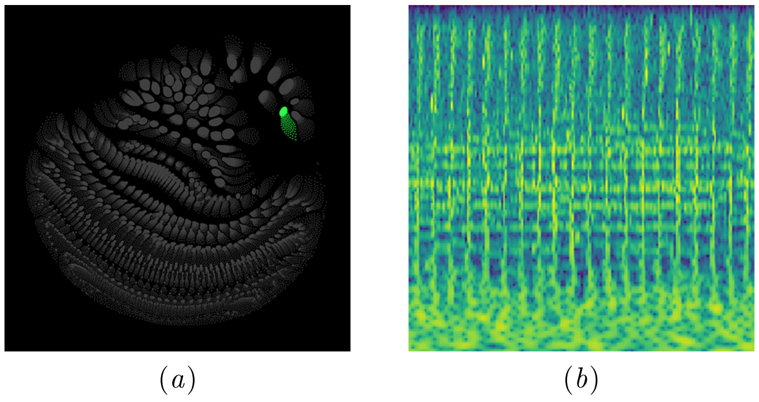

Figure 5: Input optimization for block 6 in the autoregressive model.

in the visualization (each pattern of activations yields its own distinct sounds). As origin for the optimization silence and noise can be used as well as a pre-built clip that for example dictates a certain rhythm.

All aspects of the procedure can be adjusted in real-time. For intuitive control, a GUI was developed and the most important parameters were made controllable by a MIDI controller. The setup described here is very flexible, other networks trained on different data or with different architectures can easily be employed.

6. Conclusion

We showed that contrastive predictive coding is a promising self-supervised training algorithm for learning representations of musical data and chose it as our method to construct an "artificial ear". Further work is needed to fully examine its potential as well as its limits.

The presented visualization method, besides being aesthetically interesting, allows us to explore, evaluate and understand some aspects of a working neural network more intuitively and directly. It can be adapted to most kinds of network architectures, although the number of connections in very large models might make the application difficult.

The input optimization technique can successfully generate features in the audio domain that activate certain neurons of a network. However, these results are arguably not quite as pronounced and striking as the analogous results for images. This may have to do the fact that neural network architectures, as well as training methods, are generally less matured in the field of musical audio processing. Nevertheless, fascinating sounds can be obtained that hint at the depth of those systems that is possible to explore.

References

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446, 1986.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Learning perceptually-aligned representations via adversarial robustness. *arXiv preprint arXiv:1906.00945*, 2019.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

Karl Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the maestro dataset. *arXiv preprint arXiv:1810.12247*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10 (1):37–71, 2005.

Alexander JE Kell, Daniel LK Yamins, Erica N Shook, Sam V Norman-Haignere, and Josh H McDermott. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98(3):630–644, 2018.

Einar Kjartansson. Constant q-wave propagation and attenuation. *Journal of Geophysical Research: Solid Earth*, 84(B9):4737–4748, 1979.

Robert Kozma, Roman Ilin, and Hava T Siegelmann. Evolution of abstraction across layers in deep learning neural networks. *Procedia computer science*, 144:203–213, 2018.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. 2015.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Sherjil Ozair, Corey Lynch, Yoshua Bengio, Aaron van den Oord, Sergey Levine, and Pierre Sermanet. Wasserstein dependency measure for representation learning. *arXiv preprint arXiv:1903.11780*, 2019.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79, 1999.

Loup Verlet. Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.

Chris Walshaw. A multilevel algorithm for force-directed graph-drawing. *Journal of Graph Algorithms and Applications*, 7(3):253–285, 2006.