3

Big Data Multimedia Mining: Feature Extraction Facing Volume, Velocity, and Variety

Vedhas Pandit, Shahin Amiriparian, Maximilian Schmitt, Amr Mousa and Björn Schuller

3.1 Introduction

With several hundred hours of naturalistic, in-the-wild videos and music being uploaded to the web per minute and millions of short texts being uploaded every day on social media, the *big data* era brings a plethora of opportunities yet also challenges to the field of *multimedia mining*. A modern multimedia mining system needs to be able to handle large databases with varying formats at extreme speeds. These three attributes, *volume*, *velocity* and *variety*, together define big data primarily. After a general introduction to the topic highlighting the big data challenges in terms of the three named Vs, we give an insight into traditional techniques and deep learning methodologies to cope with the scalability challenges in all these three respects. The inherent qualities of the data driven deep learning approach – which make it a promising candidate in terms of scalability – are then discussed in detail, along with a brief introduction to its constituent components and different state-of-the-art architectures. To give some insight into the actual effectiveness of the deep learning method for feature extraction, we present the latest original research results of a showcase *big data multimedia mining* task by evaluating the pretrained CNN-based feature extraction through process parallelization, providing insight into the effectiveness and high capability of the proposed approach.

The internet and smart devices today, coupled with social media and e-commerce avenues, have made data abundant, ubiquitous and far more valuable. No matter what activity one is involved in at any time of the day, whether watching TV, jogging or just stuck in traffic, each activity can create a digital trace. The upsurge in social media users (e.g., Facebook, YouTube, and Twitter), with increasingly diverse, huge amounts of content uploaded every second continuously from all over the world, has made *multimedia big data* far more relevant than ever before. This includes a huge variety of photographs, sketches, home videos, music content, live video streams, news bulletins, product reviews, reviews of local businesses and tourist places, tweets

Big Data Analytics for Large-Scale Multimedia Search, First Edition.
Edited by Stefanos Vrochidis, Benoit Huet, Edward Y. Chang, and Ioannis Kompatsiaris.

Companion website: www.wiley.com/go/vrochidis/bigdata

and posts with one's political or social views, movies, gameplays, and podcasts, to name just a few. Organizations today, from industries to political parties, rely on such data to get the low-down on current trends and assess their future strategies [1]. All sorts of users can benefit from big data multimedia mining for all kinds of novel services.

All this data comes from a vast number of sources all the time from across the globe. Data analysis and feature extraction methods need to cope with these astounding rates and quantities of data streams. Data-mining tools first need to translate the data into a native representation that can be worked on, then into a meaningful representation that can effectively be handled for further data-mining and querying, which, in most cases, is in the form of feature vectors. The feature extraction or data abstraction encapsulates the essential information in a compact form suitable for certain tasks. Because what qualifies as "suitable" or "essential" can vary from task to task, this feature extraction step is often governed by typicality of the data-mining queries subject to the data modality. More often than not, therefore, the tools also need to compute feature transformations that enhance their usability by improving the discriminative and predictive scope.

Just as the term implies, one of the most defining characteristics of multimedia big data is the enormous *volume* of data in terms of both the feature dimensions and the number of instances. This defining quality, and thus availability of such a quantity of data, becomes a game changer for businesses and mining such data through a network effect. As the number of instances or the number of features go up, however, it can become exponentially harder to make predictions with the same level of accuracy. This is one of the most commonly encountered problem in the machine learning literature, called the *curse of dimensionality*. For example, the popular distance measures used to better understand data, such as the Euclidean distance, fail when the dimensionality of the data is too large. This is because with too many co-ordinates in use there is (only) a little difference in the distances between different pairs of data samples. Irrelevant feature dimensions further add to the unreliability of this similarity or the distance measure, as each individual dimension affects the distance computation directly. Ironically, even the thousands of instances existing in 100-dimensional space is very sparse data. The dataset may appear like a bunch of random points in space with no specific underlying pattern. To have any better prediction capability, one will likely need millions of instances, thanks to the high dimensionality of the feature vectors (100s in this case). A high volume of data also necessitates higher processing memory and storage capabilities.

Also, as discussed earlier, the tools first need to translate the data into native representation, and then into its abstraction as the features, before any querying or data-mining can even begin [2]. Storing the data as native representation itself requires some non-zero time. In view of the enormous *velocity* of the data streams as discussed previously, while it is unreasonable to expect to finish even the storing of data, fast processing and encapsulation become rather a necessity in the multimedia big data context. In benchmarking studies on big data, therefore, computation time is often the prime performance measure.

The ubiquity of data generators presents us with another challenge. Because the data is gathered from a huge range of devices and sources, and features a rich diversity in terms of multimedia formats, also in part due to manifold different multimedia codecs in use these days, it is often an alphabet soup of data with a *variety* of file formats and hierarchical structures. This is especially true for multimedia data where it also features multitudes of modalities such as images, videos, audios, documents, tweets, binary system

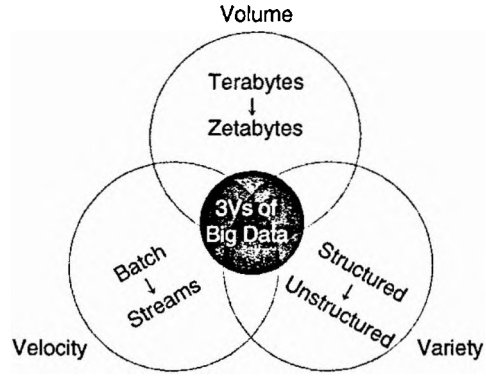


Figure 3.1 The Vs of big data.

logs, graphs, maps, overlaid live traffic or weather data and so on. Affordable new technology devices, such as Kinect, continue to introduce new modalities of data. More specifically, in the case of Kinect-like devices with 3D sensing technology, the data also consists of 3D point clusters, varying in space and in time [3].

In summary, all machine learning approaches rely heavily on the features that represent the data. A scalable data-mining approach thus requires that all of its different components are able to handle huge volume, velocity, and variety in the data (Figure 3.1) – right from the feature extraction step.

In section 3.2, we first discuss the common strategies adopted to make data-mining scalable in terms of volume and velocity, when the variety of the data has been duly considered, i.e., when the framework to represent the data in a consistent form is in place just as necessary. Next, in section 3.3, we discuss "scalability through feature engineering", which is just the process of intelligently picking the most relevant features going by the data modality and common queries. We also discuss the popular feature transformation methodologies and the contexts in which each of those are ideal. This becomes highly relevant for *model-based* approaches in particular, where the queries are attended to using explicit heuristics on the features. Section 3.4 introduces an implicit feature and representation learning paradigm, also called the *data-driven* approach, of deep learning, where the most relevant features are implicitly learnt by establishing mapping between the inputs and outputs through nonlinear functions. We argue that the inherent qualities of this model make it a great candidate for highly scalable machine learning as well as a feature extraction paradigm. A few of these qualities are (i) high flexibility in terms of the number of simultaneous outputs and the variability in terms of what each can represent, (ii) the breadth of high-level concepts it can model, e.g., the temporal and spatial correlations along with the intertwined contexts, (iii) the high modularity and integrability of the models, (iv) the wide scope for innovation through mixing and matching of various model topologies and the constituent elements, (v) the scope for velocity scalability through parallelization of the recursive and repeated elements and functions in the model hierarchy, which are also its indispensable components, such as the activation functions and kernels. We also discuss the key elements, most common architectures, and state-of-the-art methods that have evolved through this mix-and-match approach. Keeping the uninitiated reader in mind, there are detailed graphical illustrations accompanying the text to help easy intuitive understanding. We present benchmarking experiments in section 3.5 on testing; for the very first time, the runtimes of pretrained CNN-based feature

extraction on audio data, with and without process parallelization, are presented. Finally, in section 3.6 we present our concluding remarks.

3.2 Scalability through Parallelization

Due to the high volume and velocity of the data, the main challenges posed by big (multimedia) data are the data processing overheads and the added memory requirements. To meet these challenges, the data-mining approach needs to be highly scalable. The common strategies adapted for scalability fall into two categories: (i) improve the scalability of the machine learning algorithm itself using, for example, kernel approximations, parallelization of the processes or the data, or a combination of all three methods, and (ii) reduce the dimensionality of the data by generating the most compact and useful data representation possible, alleviating formidable memory and processing requirements, also called *feature engineering*.

For the first approach, for example, one of the most widely used machine learning algorithm is the support vector machine (SVM). An SVM (Figure 3.2) separates different clusters in the data using hyperplanes (linear kernel), hypersurfaces (polynomial kernel), or hyperspheres (radial basis function kernel). Being such a popular classical approach, several implementations parallelising the SVM process chain have been proposed [4–8]. Scalable implementations of several other approaches also exist, e.g., the random forest [10, 11] and or linear discriminant analysis [12, 13].

There are two main ways in which parallelization is achieved (Figure 3.3). These two parallelization techniques, discussed next, could also be combined.

3.2.1 Process Parallelization

In this approach, an algorithm is split into different smaller tasks, and these tasks are divided among the computing machines for faster processing. This way, parts of the program are executed simultaneously on different processors, and the program takes much less time to finish.

3.2.2 Data Parallelization

In this approach, the data are split into different batches. These data batches are then sent to the different processing units available, all of which house the same set of execution

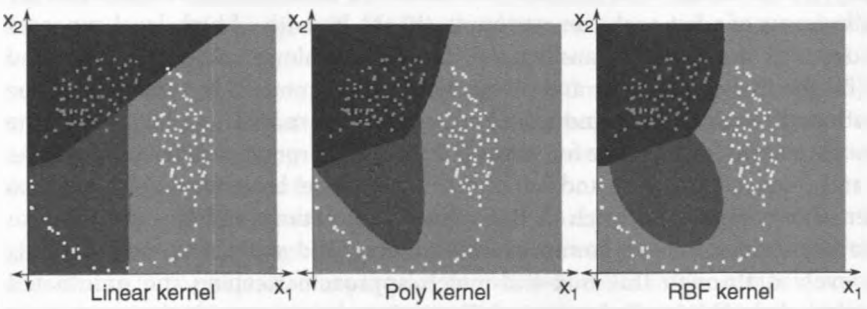


Figure 3.2 SVM kernels illustrated using web-based implementation of LibSVM [9].

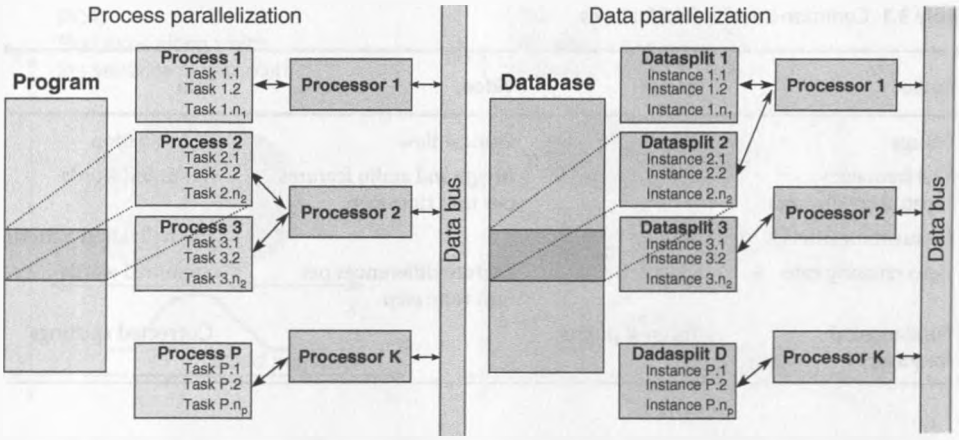


Figure 3.3 In the process parallelization scheme, a task is split into difference processes to be handled by different processing units simultaneously. In the data parallelization paradigm, the input data is split into different chunks or batches, each of which is handled by a separate processing unit independent of the other.

commands. This way, a large amount of data is processed simultaneously and sequential processing of the data is avoided. In the MapReduce programming model likewise, the data are divided into independent chunks which are processed by the mapping tasks simultaneously in a completely parallel manner. This is similar to data parallelism. These chunks then act as an input to the reducing tasks, whose prime job is to summarize the mapped information.

3.3 Scalability through Feature Engineering

Feature engineering attempts to reduce the dimensionality of the raw data through customized feature computations or through feature transformations, e.g., handcrafted feature extraction, principal component analysis, or use of pretrained autoencoders for optimal data representation.

Depending on the type of the data, and knowing the typical queries for that modality of data, the best defining attributes or features are often extracted. As an example, for an image, detecting the edges or blobs (i.e., the regions that differ in properties, e.g., colour, compared to the surrounding regions) is often of great interest and helps identify objects in the image. Similarly, for an audio data, the energy and dominant frequency of the signal together translate to the perceived loudness, subject to the equal loudness contour. The frequency domain representation of the audio provides us with information about the array of dominant frequencies, and consequently the formants and the harmonic structure. As for the speech signal, knowing the relative location of the formants helps to estimate the vowel being spoken. For video sequences, optical flow is the feature of interest that helps the relative velocities of the objects in an image to be estimated. Table 3.1 lists the most common handcrafted features particular to the common modalities of the data.

Table 3.1 Common handcrafted features.

Audio	Image	Video	Text
Energy	Edges	Optical flow	Tokenization
Mel frequency cepstral coefficients	Blobs	Image and audio features per unit time step	Stemmed words
Formant locations	Ridges		Capitalization pattern
Zero crossing rate	Corners	Feature differences per unit time step	Stemmed words
Fundamental frequency	Interest points		Corrected spellings

3.3.1 Feature Reduction through Spatial Transformations

This is done by decorrelating variables through matrix factorization (e.g., non-negative matrix factorization (NMF)), analysis of variances (e.g., principal component analysis (PCA) and linear discriminant analysis (LDA)). For example, PCA transforms observations of possibly correlated variables into a set of linearly uncorrelated variables called the *principal components*. Data is first transformed into a new co-ordinate system such that the greatest variance by some projection comes to lie on the first co-ordinate, the second greatest on the second and so on. In many cases, reconstruction using only the top few principal components is an accurate enough description of the data. PCA is purely statistical in nature, and it takes into account all of the data samples without discriminating between the classes the samples belong to. This approach is commonly known as the *unsupervised* approach. Certain other techniques use the *supervised* approach. These make use of the "class label" information identifying most discriminating attributes, in other words, the most useful features for classification tasks. LDA is one of these techniques. This method also relies on a linear transformation of the features similar to PCA, but it attempts to compute the axes that maximize the separation between multiple classes, rather than maximizing the variance across all of the data samples. The difference between the two approaches can be visualized from Figure 3.4. However, PCA and LDA are primitive techniques, they are not as useful when it comes to feature reduction on massive amounts of data. Several methods for feature selection have now been implemented that are based on variance preservation or use SVM that are aimed at data-mining on large-scale data [14–16]. The open source toolkit *openBliSSART* was the first to bring non-negative matrix factorization to GPU [17].

3.3.2 Laplacian Matrix Representation

In the simplest terms, the motivation behind algorithms like spectral clustering approaches is to take into account the adjacencies of every data point with respect to the dataset as a whole, rather than merely computing the pairwise distances across the data. Figure 3.5 presents an example case where this can be particularly useful. The two data points A and B, belonging to the same cluster, are far away from each other. The pairwise distance of the point C from B is much smaller in comparison. Yet, intuitively speaking, it makes sense to assign the points A and B to the same cluster,

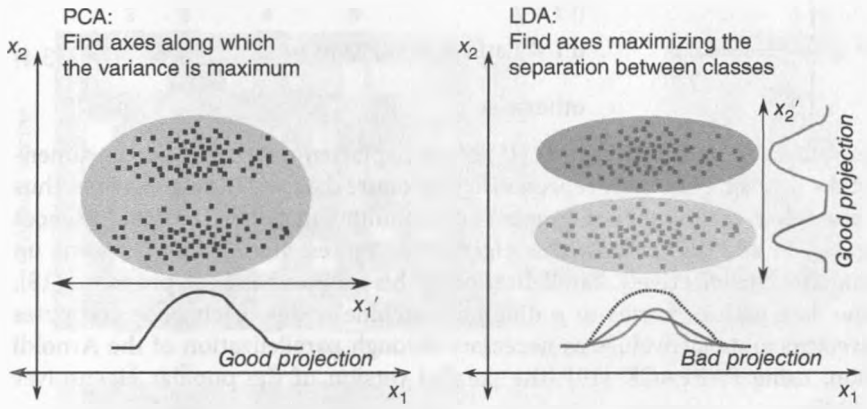


Figure 3.4 Difference between PCA and LDA feature reduction techniques, reducing the feature vector dimensionality from 2 to 1 in the example above. PCA aims to find the axes along which the variance of the data is maximum. The corresponding transformation thus generates feature dimensions corresponding to linearly uncorrelated variables. PCA, however, does not take into account the class labels, so the variance maximization alone therefore may result in a non-ideal projection (axis X_1 or X_1'). LDA aims to find axes maximizing separation between the classes (axis X_2 or X_2').

and to bin the point C into another, looking at how the two data points A and B are related or "connected" through the other data points in the feature space.

To achieve this, first a *Laplacian matrix* (L) is built using the pairwise distances between the individual data points v_i and v_j (called the *adjacency matrix* (A)) and the cluster they belong to (represented by the *degree matrix* (D)):

$$L = D - A \tag{3.1}$$

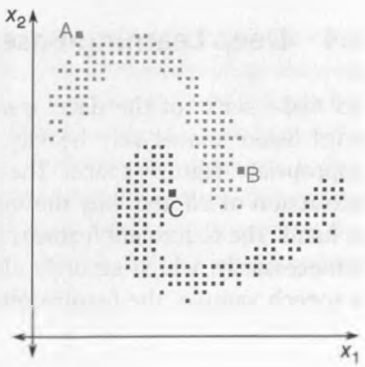
or in the symmetric normalized form:

$$L^{sym} = I - D^{-1/2} A D^{-1/2} \tag{3.2}$$

The elements of L and L^{sym} are given by following equations:

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Figure 3.5 Spectral clustering takes into account data adjacencies in addition to the pairwise distances between the data points. The points A and B are far apart, but belong to the same class. They relate to each other through other data points in their close vicinity (adjacency) and the adjacencies of those data points in iteration. While the points B and C are much closer in terms of euclidean distance, the data adjacency consideration makes it clear that they do not belong to the same class.



$$L_{i,j}^{sym} = \begin{cases} 1 & \text{if } i = j \\ \frac{-1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

The eigenvalues (λ_i) and eigenvectors (V_i) of the Laplacian matrix reduce the dimensionality of the dataset, effectively representing the entire dataset. The eigenvalues, thus obtained, also tell us the number of "graphs" or the number of connected data instances in the dataset, in addition to how this eigenvector representation linearly sums up representing the data effectively. Parallelization of this process has been proposed [18], splitting the data with n entries to p different machine nodes. Each node computes n/p eigenvectors and eigenvalues as necessary through parallelization of the Arnoldi factorization, using PARPACK [19] (the parallel version of the popular eigensolver ARPACK [20]).

3.3.3 Parallel latent Dirichlet allocation and bag of words

The latent Dirichlet allocation and the bag of words approaches are very similar, where every "document" (this can be also a non-textual sequence of audio, video, or other data) is viewed as a mixture of "topics", each defined by a multinomial distribution over a W-word vocabulary. The documents (or the data content) with similar distribution in terms of topics (feature vectors) are thus considered to be similar. The paper that first proposed this approach in 2003 [21] discusses the evolution of this approach from the classical *tf-idf*-based feature reduction technique, and its close relationship with other intermediate approaches in its evolution such as *latent semantic indexing (LSI)* and the *probabilistic LSI model* (also called the *aspect model*). Interestingly enough, essentially the same model was independently proposed in the study of population genetics in 2000 [22, 23]. Bag of words feature representation has proved to be useful for machine learning tasks on multiple modalities of data, such as large-scale image mining [24], text data [25], audio [26], and videos [27]. A scalable implementation of this approach was proposed in 2011 [28] through both data and process parallelization, building on some of the previous work in this domain [29, 30]. Recently, our group released an open-source toolkit, called *openXBOW*, that can be easily interfaced with multiple data modalities to generate a summarized crossmodal bag of words representation [31] (Figure 3.6).

3.4 Deep Learning-Based Feature Learning

To make sense of the data, traditional machine learning approaches (including the ones listed above) rely heavily on the representation of the available data in the appropriate feature space. The feature extraction step therefore needs to address extraction of *all* and *only* the most relevant features given the machine learning task at hand. The redundant features simply add to the dimensionality of the feature vectors unnecessarily, which severely affects their discriminating power. For example, given a speech sample, the feature sets necessary for automatic speech recognition (ASR),

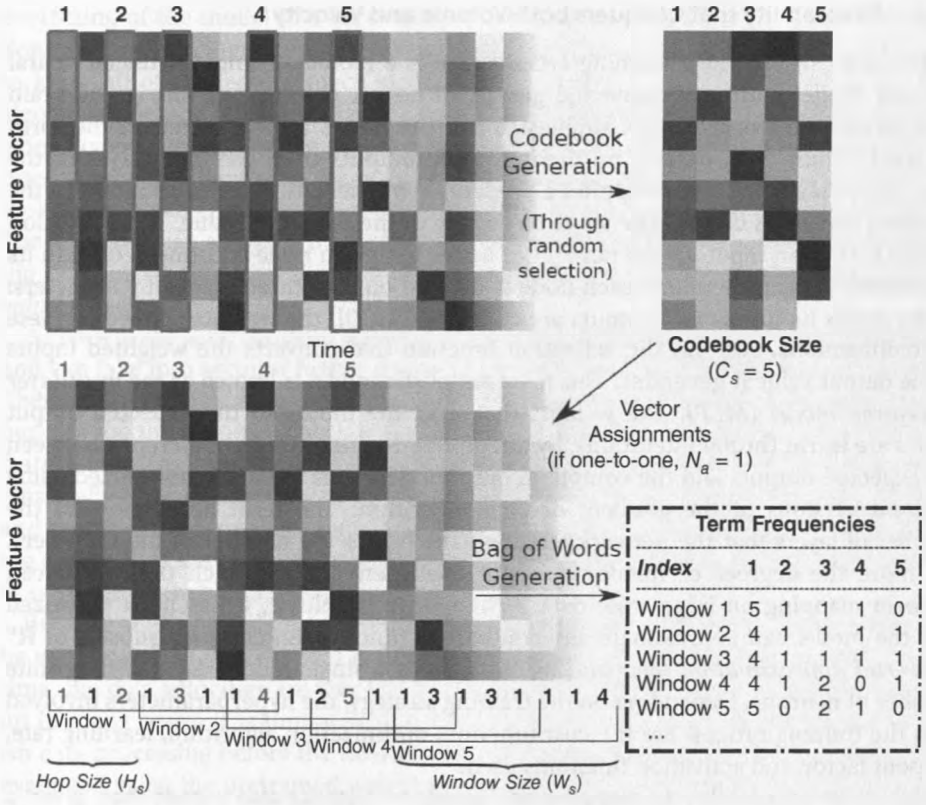


Figure 3.6 Bag of words approach. When the size of the moving window matches the size of the incoming input (or the document) itself, i.e., when the entire incoming input is represented in terms of the counts of the types of features (or the words) it exhibits, the approach closely resembles the latent Dirichlet allocation representation.

speaker/gender identification, and emotion recognition are not identical. While speaker identification is often solved by extraction of the universal background model (UBM) and i-vectors, the statistics associated with fundamental frequency (F_0) are popular for gender identification. The more complex paralinguistic tasks might entail extraction of thousands of features [32–35].

Interestingly enough, a new approach to feature extraction has evolved that attempts to *learn* what features are most useful when given a task. This is called the *deep learning approach*, and it uses a network of cells similar to neurons in the brain. In addition to big data, this deep learning has also been a buzzword in both the commercial world and the research community in recent times. The beauty of the deep learning approach is that it abstractifies the feature extraction step as part of the training phase itself. Depending on the task at hand, this artificial neural network-based model attempts to reorient the intermediate computations (also therefore the "features" in the abstract sense) to best map the given inputs to the single or array of desired outputs.

3.4.1 Adaptability that Conquers both Volume and Velocity

At the heart of any deep learning architecture is a biology-inspired artificial neural network model with interconnected groups of nodes, akin to neurons in the brain as discussed previously. In its simplest form, the nodes are organized in the form of layers (Figure 3.7), passing on the individual outputs from the input layer to the final output layer. Each node applies a predefined mathematical transformation to the weighted sum of its inputs to generate an output, or the activation value. This activation in turn acts as an input for the next set of nodes the given node is connected to. In its rudimentary form, therefore, each node is defined only by three types of parameters: (i) the nodes its inputs and outputs are connected to, (ii) the weights applied to these interconnections, and (iii) the activation function that converts the weighted inputs to the output value it generates. This most simplistic model is known as the *multilayer perceptron model (MLP).* The weights that map the inputs to the expected output values are learnt through iterations, by attempting to minimize the difference between the expected outputs and the computed outputs. This cost is mostly minimized using different versions of the gradient descent algorithm. The term *deep* refers to the number of layers that the network features. The higher the number of hidden layers, the more the degrees of freedom (weight coefficients) with which the model can perform mapping on large-scale data. While fairly simplistic, it has been theorized that the model can approximate any continuous function on compact subsets of $\mathbb{R}^n$ (*universal approximation theorem* [36]) using only a single hidden layer with a finite number of neurons. Depending on the training strategy, the hyperparameters involved with the training process are the cost function, optimization algorithm, learning rate, dropout factor, and activation functions used.

Such great adaptability in mapping any inputs to outputs also entails that, given a very small set of inputs and outputs with the cost function, there likely are multiple possibilities for the "optimal" mapping. Thus, achieving the most generalizable and robust mapping necessitates also the use of a large number of training samples in order to avoid

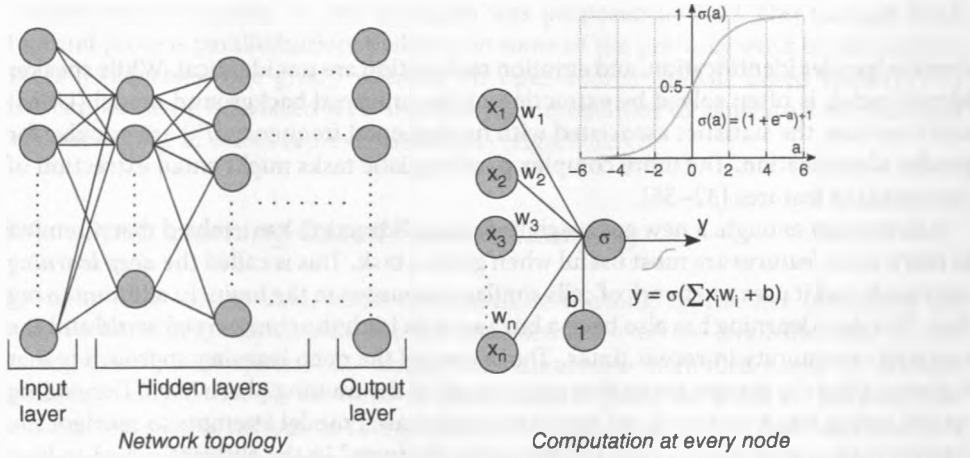


Figure 3.7 The multilayer perceptron model. For every node, the output y is computed by applying a functional transformation $\sigma(a)$ to the weighted sum of its inputs, where weights are denoted by w_i corresponding to input values x_i.

overfitting of the model to a very small set of instances. Also, because this training is done iteratively or incrementally, the training step inherently does not require the model to know all of the feature space in its entirety at once. This is opposite to SVM-like algorithms, which often rely heavily on the data clusters present in the feature space when generating the discriminating hyper-surfaces. Therefore, while large-scale data is often a problem due to the memory and computation bandwidth available, it is often looked on as the great advantage when it comes to the neural network-based models. These models can also afford to split the data into smaller chunks without compromising as much on the generalization of the mapping, and thus deal more easily with the *volume* issue.

As we will see later, the deep learning model can handle the variety of data modalities, and can take into account higher concepts such as time and the temporal correlations. Therefore, it is possible to feed in not only scalar values of the data to the input nodes, but also the the streams of scalars (i.e., feature vectors), streams of feature vectors (i.e., feature matrices), and streams of feature matrices, and so on. This hierarchical mathematically generalizing structure is formally known as a tensor, where scalars, vectors, and matrices are tensors of rank 0, 1, 2 and so on. The uniqueness of the deep learning approach is that it is perhaps the only approach to date that can work with tensors of rank higher than 2 as its inputs.

Because these models can easily reorient and adapt to incoming inputs, they can be used in online systems that are continuously fed with large streams of data in real time. Big data velocities are way too stupendous for even the storage systems to catch up in many cases; it is impractical to expect the machine learning algorithms to finish data processing before the next samples show up. Because the incoming input gets evaluated using the pretrained weight parameters with simple mathematical functions alone (without looking at the subspace of the dataset, or the decision trees, like in other algorithms), the feature-set extraction and the prediction task can be performed at a much faster rate. Also, because the implicit feature-set extraction and prediction are both mathematical functions involving repeated use of subfunctions, such as sigmoid or tanh on the matrices or tensors as their argument, advanced matrix manipulation techniques and parallelization can in theory be applied for faster throughput. The developers of one of the most popular and influential toolkits today, called *Tensorflow*, have released a white paper recently which briefly mentions their plans on introducing just-in-time (JIT) compilations of the subgraphs [37].

The autoencoders present a special case of deep neural networks, where the output is identical to the input and therefore a different/compressed (or rarely also expanded) representation of the data is learnt intrinsically. Because the inputs always map to themselves, the intermediate layers (generally smaller in dimension compared to the input dimension in the case of a compression autoencoder) represent their compact or "noise-free" representation. To make the model (and consequently the reduced representation) more robust against the noise and spurious inputs, the network is often trained using original, clean input as its expected output, while feeding in the noisy version of the input to the network (Figure 3.8). This class of autoencoders is called denoizing autoencoders. Because they map inputs to themselves through mostly nonlinear transformations, they are often said to perform "implicit learning" of the data. Such compressed encoding of the input is often referred as the "bottleneck" representation. The autoencoders can be hierarchically stacked by making the encoded

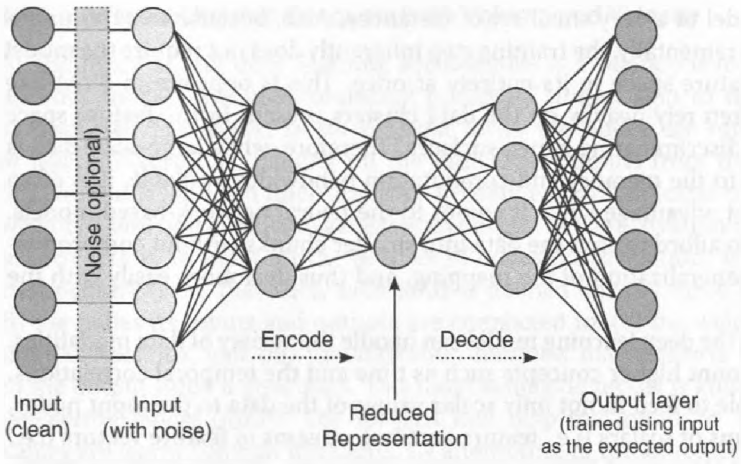


Figure 3.8 The autoencoder (compression encoder). The hidden layers represent the essence of the data provided, a meaningful compact representation of the inputs, also called the "bottleneck" representation.

representation run through a compressing autoencoder, implicitly learning the encoded representation. This approach has been useful in domain-adaptive sentiment recognition from speech [38], content-based image retrieval [39], estimating visual tracking for videos [40], paraphrase detection in texts [41], and a myriad of further tasks.

3.4.2 Convolutional Neural Networks

While the simple neural networks manage to map inputs to their outputs, such a network often features a fully connected graph (albeit not in both directions), which completely misses out on important dependencies or relationships between the node groups. To perform an image recognition task, for example, a simplistic MLP model might use a flattened single dimensional array generated from 2D pixel values, without any anticipatory consideration for the spatial relationships existing in the data. We know from experience, however, that the edges in an image are often defined by the differences and deviations between the intensities of the neighbouring pixels. Their spatial relationships define how slanted the edges are with respect to the horizontal or vertical axes. It makes sense to assign high magnitude weights (irrespective of the sign) to the neighbouring pixels rather than those very far away. A carefully weighted sum of the pixel values, with weights that represent a certain preselected edge orientation (called the *receptive field kernel*), tells us whether or not the pixel neighbourhood features a preselected image pattern, e.g., an edge, or a blob (Figure 3.9). High correlation between the pixel neighbourhood and the receptive field translates to a high activation value. Different receptive fields moving across the image through fixed steps (called *strides*) in both dimensions can effectively localize different edges and patterns in the image. Understanding of the spatial relationships between these patterns, and consequently localization of the high and low activation values, effectively translates to nothing but an object recognition task itself. Typically, the "pooling layers" are also introduced in between the layers. The pooling layers reduce the dimensionality of the input matrix size, while preserving the salient information obtained in the earlier computations

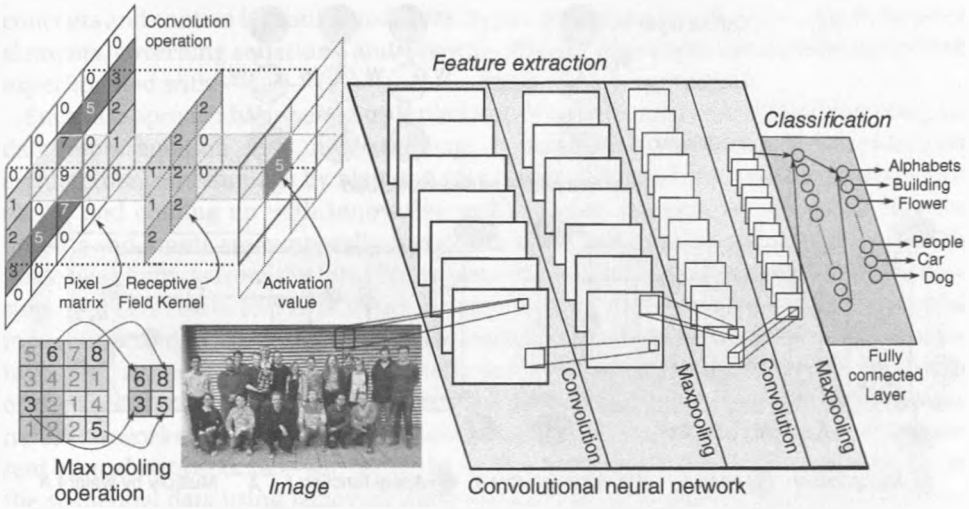


Figure 3.9 In a convolution operation, the receptive field kernel (size: 3 × 3) is multiplied element-wise with the subset of a pixel matrix of the same size, and these weighted values are summed together to compute the activation value. A max pooling operation reduces the dimensionality by computing the maximum value in the kernel of a given size (size: 2 × 2).

through a non-linear transformation, such as a summing or a maximizing function. The typical example of a pooling layer is the max pooling layer that applies a *max* operation for the stated kernel or window. Figure 3.9 features a max pooling layer of a size 2 × 2 kernel and a convolution kernel of size 3 × 3 for illustration purposes. The feature representation in the form of activations by the first few convolutional layers often translates to the likelihood of certain pixel patterns being present in the image. This, in some sense, is similar to the bag of words approach. The spatial correlation between these patterns (captured using the convolutional kernel) translates to detection of higher order patterns. Because these weights get trained, this allows for much more complex mappings and feature manipulations beyond the standard bag of words approach.

3.4.3 Recurrent Neural Networks

The simple multilayer perceptron model fails to take into account time-related dependencies and relationships existing in the data, which is of utmost importance when it comes to usual multimedia data that is of a sequential form, e.g., an audio clip, frames of a video, or even textual data. In these cases, the expected output is dependent not only on the current input, but also on the sequence of preceding and potentially following inputs. To model this kind of relationship, the outputs of the hidden layer are often connected back to its own input. In this way, each node in the layer computes a weighted sum of its inputs coming from a previous layer and the last output it generated. The inputs from the previous time step, therefore, also affect the activation computation in the current step. In a bidirectional architecture, the future observations are also used.

Such a network that uses simple perceptrons as its nodes, computing mere non-linear transformation of the weighted inputs, fails to model long-term dependencies. To provide the network with the desired capability, alternate topologies for the nodes

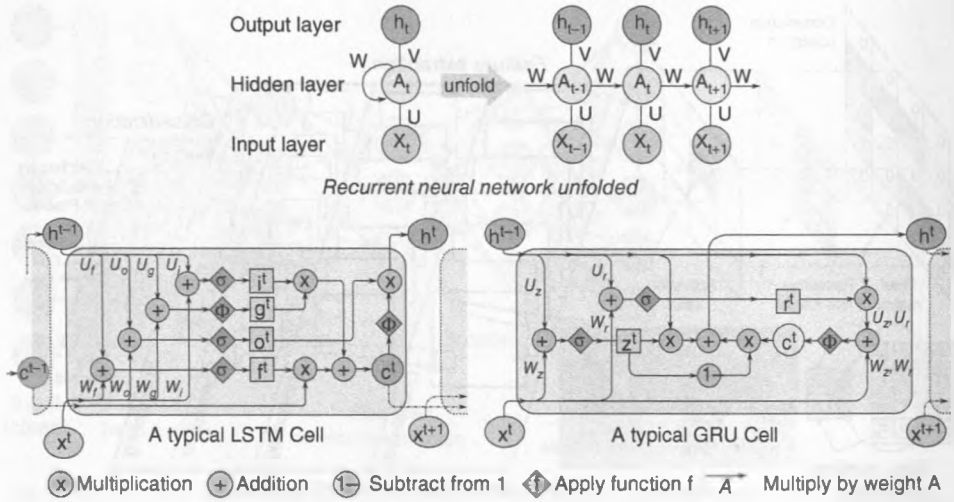


Figure 3.10 In a recurrent neural network, output from a node is fed back to itself to make the past inputs affect the outputs at the next time step. A simple perceptron, however, fails to capture the long-term dependencies. The modified cells with internal memory state (e.g., LSTM and GRU cells) are used to alleviate this problem.

were proposed which feature an internal memory state. The different node topologies only differ in the way this internal memory state is calculated, and the way this state is used to compute the outputs in the current and the next time steps. Figure 3.10 illustrates two of the most popular topologies in use today, namely *long short-term memory (LSTM)* cells and *gated recurrent unit (GRU)* cells. By mapping the temporal relationship of the samples in the sequential data with respect to one another, and also with the respect to the sequential (or the summarized static) output values through reiteration and re-usage of the mapping functions, we get the more compact and most essential representation of the input sequence in terms of far fewer parameters when using recurrent neural networks. The open source toolkit *CURRENNT* was the first one to have GPU implementation of the bidirectional LSTM (BLSTM) [42].

3.4.4 Modular Approach to Scalability

The deep learning approach is characteristically modular and therefore multiplicatively scalable. One can mix and match different numbers and types of layers (e.g., fully connected, convolutional, recurrent, max pooling) with different numbers and types of nodes in each layer (e.g., perceptron, LSTM cell, GRU cell) with different activation functions (e.g., sigmoid, tanh, linear) with a variety of interconnections (densely connected, sparsely connected neural network), also deciding how the output relates to the inputs (single or multiple class labels, single or multiple regression values, or simply de-noised inputs) when training the network. This presents a multitude of possibilities in terms of the network architecture itself, and what all it can possibly model. Depending on how complex the problem is, and/or how sparse the data are, and what higher level

concepts and context it should model, the hyper-parameters such as the type of network elements, governing equations, and quantity of layers and nodes can be determined and experimented with.

Such an approach has opened up a plethora of opportunities for research in compact data representation and understanding. It has spawned several new interesting architectures and findings by blending ideas from different basic topologies discussed earlier and coming up with innovative architectures. Because in sequential data the outputs and inputs are temporally correlated, some of the early publications on LSTMs made use of inputs from the future time steps to compute the output at the current time step. This gave rise to *BLSTMs*, which is popular today. These have been especially useful in helping computing systems better understand sequential media such as speech, for tasks such as speech overlap detection, language, and acoustic modeling [43, 44]. Some of the recent publications propose network architectures in which the convolutional neural networks model the sequential data through what are called *convolutional recurrent neural networks* [45–48]. Some have also attempted compact representation of the sequential data using *recurrent autoencoders* [49]. A variant of such autoencoders, called variational autoencoders, employ a special kind of cost function that forces the bottleneck representation to be as close as possible to a Gaussian distribution [50]. Recently, drawing inspiration from their earlier work on image generation, called PixelCNN [51], researchers from *DeepMind* came up with a generative model for speech synthesis called *WaveNet*, using what they called *causal convolutional layers* [52]. Through training using real-life examples, the network is able to generate samples which when put together resemble natural speech. Because one can also have generative models beyond the predicitive ones, using deep learning networks, recently researchers mixed the two, generating an interesting unsupervised machine learning model called *generative adversarial networks (GANs)* [53]. In this approach, the two networks, one generative and the other predictive, compete against each another in a zero-sum game framework. The basic idea is that the generative network is trained to generate synthesized data instances using the training instances from a latent variable space, while the predictive network is simultaneously trained to differentiate between the synthesized instances from the true instances. At the other end of the spectrum, in terms of their simplicity, are *extreme learning machines (ELMs)* in which the inputs to hidden layer weights are randomly assigned and are never updated [54]. In ELMs, only the hidden layer to output mapping is learnt through iterations. ELM training thus becomes essentially a linear learning problem [55], significantly reducing the computational burden. While controversial [56–58], these simplistic models seem to work on complex tasks such as hand-written character recognition [59] or excitation localization for the snore sounds [60]. Another variant of neural networks, called the residual network, makes use of "shortcut connections" to skip over a few layers to feed the signal to the desired layer for summation. Thus, instead of directly training the network for the mappings desired (say, $[H(x)]$), the residual network instead is trained to learn the mapping between the input and the desired output that is offset by the input itself (i.e., $[H(x) - x]$). This modification in the topology has been shown to alleviate the problem of *degradation*, which is the saturation in accuracy with an increase in the number of layers or the depth of a network [61].

3.5 Benchmark Studies

The deep learning approach can effectively and intelligently process various kinds of data. As discussed, it can easily be customized to account for temporal dependencies and the contexts, which that are both intrinsic and vital when it comes to the sequential data, e.g., raw audio clippings, tweets, and news articles. One can also process spatial information, e.g., photographs, plots, and maps. Even data where the spatial and sequential contexts are highly intertwined, e.g., videos, console games, and live traffic maps, can be dealt with effectively using deep learning approaches through little adaptations in the network and the cells in use. The data can therefore be used in almost its original form (called *end-to-end learning*), without having to introduce any customized feature extraction step. The classical approach requires every data sample to be represented in feature vector form, and overall data as a 2D matrix (i.e., a rank 2 tensor). Because the neural network based models can handle higher level concepts (such as the time) effectively, the data is often represented as a tensor of rank even greater than 2. By extension therefore, the data learning process is inherently scalable, not only in terms of the number of features in use or the number of samples, but also the modalities and rank of the tensor it can handle at any given time. We discuss previous benchmarking studies on deep learning in this section.

Due to its huge potential and the consequent drive from both the industry and the research community, there is now huge growth in the number of deep learning toolkits available. Some of these were once the in-house frameworks for industry giants like Google. The most popular frameworks are Caffe, 2018, CURRENNT [42], Microsoft Cognitive Toolkit [62], Tensorflow [37], Theano [63], and Torch [64]. Some of these, like Keras [65], come with an API capable of running on top of other frameworks, e.g., Theano and Tensorflow.

Benchmark comparisons between these deep learning frameworks have been conducted in the past. The computational performance of the single framework depends highly on the available hardware architecture. One of the most critical parameters is the type of processors in use, i.e., whether *central processing units (CPUs)* or the *graphics processing units (GPUs)*, or, in the near future, *neuromorphic processing units (NPUs)*. The latter provide a much larger degree of parallelization and are therefore suitable for training deep neural networks. As the actual performance also depends on the employed version of the corresponding software tools and new frameworks are published on a regular basis, the reader is referred to online resources to catch up on the latest benchmarks.

Whereas these deep learning frameworks are well-studied in terms of their performance in machine learning tasks of all kinds, this is not the case for the feature-extraction step. In the following sections, we provide experiments and results for feature extraction via deep learning. For our benchmarking studies, we run feature extraction on the audio data using pretrained convolutional neural networks.

3.5.1 Dataset

We use the TUT Acoustic Scenes 2017 dataset, currently in use for the DCASE2017 Acoustic scene classification challenge, that was recorded in Finland by Tampere University of Technology. The dataset consists of 4680 samples recorded in 15 acoustic

scenes such as the forest, library, metro station, and restaurant indoors. At each recording location, a 3–5 minute high-quality audio was captured which was then split into 10-second segments. Each audio clip was recorded using 44.1 kHz sampling rate and 24 bit resolution. While this is by no means big data, we upsampled the data through repetition to a high data volume to showcase the principles.

3.5.2 Spectrogram Creation

When it comes to working with audio data, the dynamics in the frequency domain are often a good indicator. Therefore, features based on the frequency domain representation of the audio signal are almost always extracted for any machine learning tasks, e.g., Mel-frequency cepstral coefficients (MFCC), formant locations, or fundamental frequency (F_0). A varying spectrum of signal over time is often represented using a spectrogram, computed using short-time Fourier transform (STFT) or the wavelet transform. The horizontal axis represents time, the vertical axis represents frequency, and the magnitude or intensity of the continuous-time complex exponential of a particular frequency at a particular time is represented by the color or intensity of the point at that location. As long as one chooses the 'right' frame and hop size, taking into account the time frequency uncertainty principle for a given task, the spectrogram representation captures all of the quintessential information necessary. Instead of the raw audio therefore, we used the spectrogram representation as our input in this example. Summarizing, we translated the audio inputs into a format that can be best processed by the pretrained CNNs we aim to use.

Based on our previous experiments [66], we used the Hanning window of size 256 samples and traversed the audio data with a hop of 128 samples. We computed the power spectral density in decibels using the Python package numpy [67] and spectrogram representation using the matplotlib [68] package, in the perceptually uniform sequential colour mapping called *Viridis*. Viridis uses blue to represent the low-range values, and green and yellow for the mid-range and high-range values, respectively. As in our previous experiments, we got rid of the axes and the margins presenting redundant information that was common across all images by cropping programmatically. The original spectrogram images were then 387 × 387 pixels, which we rescaled to 227 × 227 pixels and 224 × 224 pixels, to comply with the AlexNet and VGG19 requirements, respectively. Figure 3.11 contains a few of the spectrograms we generated through this process from audios from four different classes. As can be seen, the spectrograms differ a lot visually, so much so that some clear distinctions can be made even with the human eye. The CNNs pretrained for image classification tasks in particular are, therefore, expected to perform well on this data, extracting the summarized feature representation with good discriminative scope.

3.5.3 CNN-Based Feature Extraction

Instead of explicitly computing the higher level audio features, e.g., F_0 and MFCC, through a dedicated feature extractor, we fed the spectrograms to pretrained CNNs that have been proven to work in image classification tasks. More specifically, we extracted the features using the pretrained AlexNet and VGG19 CNN (see chapter 1, Refs. 11, 23). The AlexNet was trained using ImageNet data, which featured more than

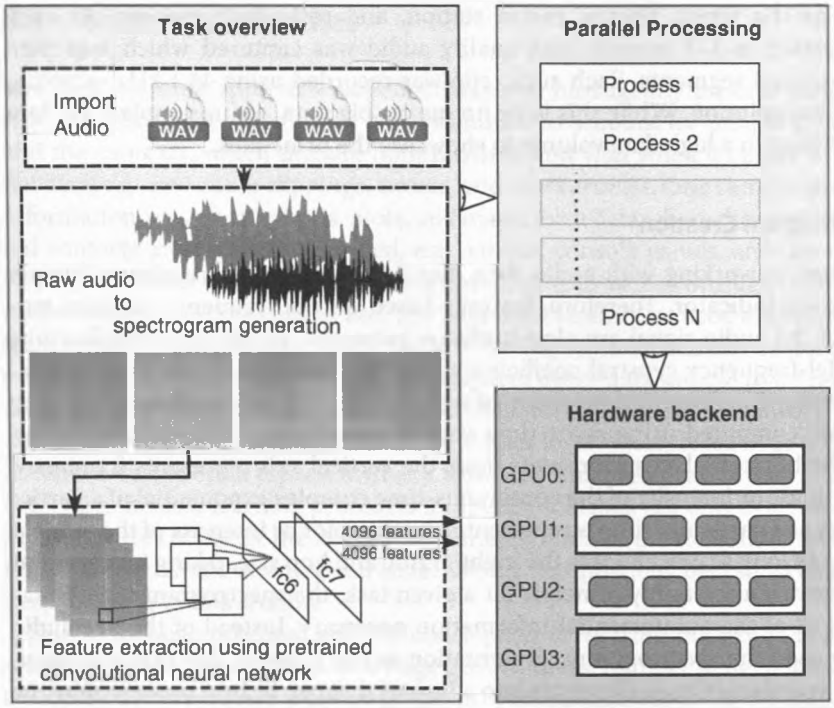


Figure 3.11 Overview of the system with CNN-based feature extraction through parallelization among GPU cores. In the example, spectrograms are generated from the audio data and plotted using the Python matplotlib package. The preprocessed plots are used as an input to the pretrained CNNs. The activations of fully connected layers are then extracted as large deep spectrum feature vectors, using the Caffe framework. The overall task is split into several processes and is handled by four GPU cores. Another thread writes the feature vectors into an output file.

15 million annotated everyday images from over 22 thousand categories, and we tested efficacy of this network in classifying spectrograms likewise [66, 69].

Both AlexNet and VGGNet provide us with 4096 features per spectrogram. This is huge data compression in itself, considering that every audio clip consists of close to 4 million data points (44 100 Hz × 10 seconds), and the sequential structure of the data points is also very critical. While such compression necessitates spectrogram generation first, this intermediate step is not a severe bottleneck.

3.5.4 Structure of the CNNs

Both AlexNet and VGG19 use a combination of convolutional layers, max pooling, fully connected layers and rectified linear units as the activation functions (see chapter 1, ref. [26]). AlexNet consists of five convolutional layers, followed by three fully connected layers. Softmax is applied to the last layer to perform 1000-way classification. VGG19 consists of 19 layers that are grouped in five stacks of convolutional layers with max pooling. Another key difference between the two networks is that VGG19 uses only 3 × 3 kernels across all its convolutional layers, while AlexNet uses varying kernel sizes. Table 3.2 summarizes the two network architectures for comparison.

Table 3.2 Overview of the architectures of the two CNNs used for the extraction of spectral features, AlexNet, and VGG19. *ch* stands for channels.

AlexNet	VGG19
Input = RGB image	
size: 227 × 227 pixels	size: 224 × 224 pixels
1 × Convolution size: 11; ch: 96; stride: 4	2 × Convolution size: 3; ch: 64; stride: 1
Max pooling	
1 × Convolution size: 5; ch: 256	2 × Convolution size: 3; ch: 128
Max pooling	
1 × Convolution size: 3; ch: 384	4 × Convolution size: 3; ch: 256
	Max pooling
1 × Convolution size: 3; ch: 384	4 × Convolution size: 3; ch: 512
	Max pooling
1 × Convolution size: 3; ch: 256	4 × Convolution size: 3; ch: 512
Max pooling	
Fully connected FC6 layer, 4096 neurons	
Fully connected FC7 layer, 4096 neurons	
Fully connected 1000 neurons	
Output = probabilities for 1000 object classes through softmax	

Once the spectrogram plots are forwarded through the pretrained networks, the activations from the neurons on the first or second fully connected layers (called *fc6* and *fc7*) are extracted as the feature vectors. The resulting feature vector thus presents 4096 attributes, one for every neuron in the CNN's fully connected layer. These can then be passed on to either the traditional or deep learning techniques to perform automatic scene event classification, similar to previous studies on audio classification through spectrogram image classification [66, 69].

3.5.5 Process Parallelization

We use the Caffe framework to build the CNN models and for process parallelization (see chapter 1, ref. [43]). We tested the run times without and with process parallelization (six processes). The GPU in use was a GeForce GTX Titan X (GPU clock rate: 1.06 GHz). The program run includes importing the audio data, creation and initialization of the pretrained convolutional neural network, generation of the cropped and resized spectrograms, forward pass computations through the pretrained network, and writing of the 4096 output features per audio input to an output csv file. The results are tabulated in Table 3.3. To measure the run time for experiments involving more samples than what we actually have, we upsampled through repetition

the audio instances. Because we were interested in benchmarking for feature extraction process, and how it scales with process parallelization, we did not run an evaluation of the fully connected classifier section of the network outputting class probabilities (and thus, we do not present accuracy results), avoiding additional process overhead.

3.5.6 Results

VGG19 has a higher number of layers compared to AlexNet (i.e., 19 compared to 8). Just as expected, therefore, it takes longer to finish VGG19-based feature extraction, as can be observed from Table 3.3 and Figure 3.12. Because there is a higher number of processes, speed up due to process parallelization is more visible for VGG19 than for AlexNet. On average, the feature extraction process is twice as fast for AlexNet with six processes run in parallel, while it is 2.6 times as fast for VGG19. Without parallelization, the degradation in data processing rate (average number of samples processed per second) is a lot more severe, dropping from almost 9 samples/second to about 5 samples/second. With parallelization, however, data processing rates are not as badly affected with a higher number of samples. With a very high number of samples, this gain also translates to huge savings in the total computation time. The run times stated here are for the parallelization of the algorithm where raw audio file to feature vector conversion is done sample by sample in iteration. Feature extraction could be further expedited by implementing data parallelization, by splitting the data into different chunks, with each program run processing a separate batch of data in parallel.

We used our own tookit called *auDeep* for this study. It can be downloaded from the repository located at https://github.com/auDeep/auDeep/.

Table 3.3 Convolutional neural network speed up through process parallelization.

CNN	Number of samples	Run time (seconds)		Average samples/second		Speed up
		Single process	Six Processes	Single process	Six Processes	
AlexNet	500	55	27	9.09	18.52	2.04
	1 000	102	70	9.8	14.29	1.46
	5 000	520	340	9.62	14.71	1.53
	10 000	1 684	692	5.94	14.45	2.43
	25 000	4 424	1 782	5.65	14.03	2.48
	50 000	9 352	3 518	5.35	14.21	2.66
VGG19	500	91	37	5.49	13.51	2.46
	1 000	180	62	5.56	16.13	2.90
	5 000	921	343	5.43	14.58	2.69
	10 000	1 919	725	5.21	13.79	2.65
	25 000	4 900	2 218	5.10	11.27	2.21
	50 000	10 203	3 646	4.90	13.71	2.80

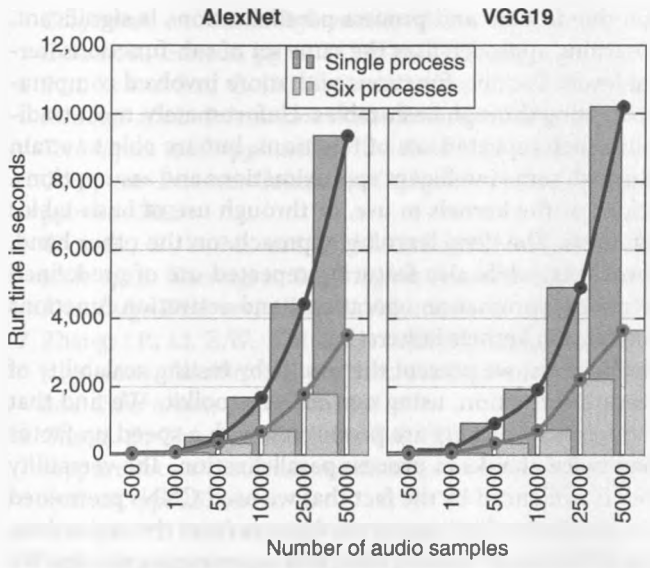


Figure 3.12 Convolutional neural network speed up through process parallelization (equal to that of Table 3.3).

3.6 Closing Remarks

Mining on big data multimedia content is a challenge in itself, mainly because of the three defining attributes of big data (volume, velocity and variety) and the multimodal nature of the multimedia. This makes it mandatory for the analytical tools to attain scalability in all three respects right from the feature extraction step. With limitations on how much hardware and software technology can scale through miniaturization and parallelization-like strategies, it comes down to inherent qualities of the competing data-mining methods as to which ones will survive the test of time.

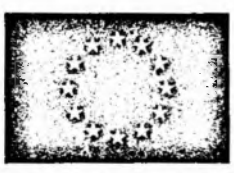
The machine learning approaches rely heavily on the representation of the data in the feature space, with *only* the relevant discriminative features retrained and *all* of the redundant or irrelevant features removed. The redundant and irrelevant features are likely to meddle with the predictive ability of the model, adding to processing overheads. Unlike traditional methods, the deep learning approach is able to learn the most relevant features, or even the feature transforms implicitly, which makes it a promising candidate for scalability in terms of both *volume* and *variety*. Many traditional approaches do not take into account the sequential structure of the data (intrinsic to some of the multimedia data, e.g., audio, live maps, videos) or the spatial relationships within the data structure (e.g., in images, plots). The deep learning approach can model temporal, spatial, and intertwined correlations as useful features. In theory, it can also take in the data in its pure raw form, without an explicit feature extraction step. These factors put the approach at much greater advantage in terms of what concepts and relationships it can model, adding much more to its potential scalability in terms of variety. The inherent modularity of the approach only adds further to the scope for innovations through a mixing and matching approach, and the versatility of the models.

In terms of *velocity*, the gain due to data and process parallelizations is significant, especially when the machine learning approach uses the same set of sub-functions iteratively at different hierarchical levels. For sub-functions with more involved computations, one can speed up the processing through hash-tables. Unfortunately, most traditional approaches do not feature such repeated use of functions, but are able to attain speed up in implementation through some intelligent approximations and assumptions, in terms of the matrix operations or the kernels in use, or through use of hash-tables for the most frequent computations. The deep learning approach, on the other hand, can make use of all these approaches, while also featuring repeated use of predefined element-wise matrix multiplications, summation operations, and activation functions like sigmoid, tanh, and the convolution kernels in iteration.

In the context of multimedia big data, we present the results by testing scalability of the pretrained CNN-based feature extraction, using our *auDeep* toolkit. We find that the results for the scalability in terms of *velocity* are promising, with a speed up factor that is almost always more than twice, thanks to process parallelization. The versatility of the deep learning framework is evidenced by the fact that we used CNNs pretrained on image classification tasks to obtain the deep spectrum features from the audio data, which have also proven to be useful for audio classification task in our earlier studies. We intend to achieve further speed up through data parallelization and just-in-time compilation of the reused functions in our future experiments, reducing the computation times further by great margins.

Likewise, the current investment trends of the industry giants, the growth in big data, data economy, and the growing competition within the deep learning frameworks unequivocally imply deep learning is the future for big data analytics.

3.7 Acknowledgements



The research leading to these results received funding from the EU's Horizon 2020 Programme through the Innovative Action No. 645094 (SEWA) and from the German Federal Ministry of Education, Science, Research and Technology (BMBF) under the grant agreement no. 16SV7213 (EmotAsS). We further thank the NVIDIA Corporation for their support of this research by Tesla K40-type GPU donation.

References

1 Mayer-Schönberger, V. and Cukier, K. (2013) *Big data: A revolution that will transform how we live, work, and think*, Houghton Mifflin Harcourt.
2 Madden, S. (2012) From databases to big data. *IEEE Internet Computing*, 16 (3), 4–6.
3 Ioannidou, A., Chatzilari, E., Nikolopoulos, S., and Kompatsiaris, I. (2017) Deep learning advances in computer vision with D data: A survey. *ACM Computing Surveys*, 50 (2), 20.
4 Graf, H.P., Cosatto, E., Bottou, L., Durdanovic, I., and Vapnik, V. (2004) Parallel support vector machines: The cascade SVM, *Proceedings of the 17th International*

Conference on Neural Information Processing Systems (NIPS '04), Vancouver, British Columbia, Canada, pp. 521–528, MIT Press.

5 Sun, Z. and Fox, G. (2012) Study on parallel SVM based on MapReduce, in *Proceedings of the International Conference Parallel and Distribution Processing Techniques and Applications* The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), pp. 495–502.

6 Caruana, G., Li, M., and Qi, M. (2011) A MapReduce based parallel SVM for large scale spam filtering, in *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Volume 4, pp. 2659–2662, IEEE.

7 Zhang, J.P., Li, Z.W., and Yang, J. (2005) A parallel SVM training algorithm on large-scale classification problems, in *International Conference on Machine Learning and Cybernetics*, pp. 1637–1641.

8 Chang, E., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., and Cui, H. (2008) Parallelizing support vector machines on distributed computers, in *Neural Information Processing Systems*, pp. 257–264.

9 Chang, C.C. and Lin, C.J. (2011) LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2 (3), 27:1–27:27.

10 Mitchell, L., Sloan, T.M., Mewissen, M., Ghazal, P., Forster, T., Piotrowski, M., and Trew, A.S. (2011) A parallel random forest classifier for R, in *Proceedings of the 2nd International Workshop on Emerging Computational Methods for the Life Sciences*, ACM, pp. 1–6.

11 Wright, M.N. and Ziegler, A. (2015) ranger: A fast implementation of random forests for high dimensional data in C++ and R. *arXiv preprint:1508.04409*.

12 Pang, S., Ozawa, S., and Kasabov, N. (2005) Incremental linear discriminant analysis for classification of data streams. *IEEE Transactions on Systems, Man, Cybernetics, and Systems, Part B (Cybernetics)*, 35 (5), 905–914.

13 Hubert, M. and Van Driessen, K. (2004) Fast and robust discriminant analysis. *Computational Statistics and Data Analysis*, 45 (2), 301–320.

14 Pudil, P., Novovičová, J., and Kittler, J. (1994) Floating search methods in feature selection. *Pattern Recognition Letters*, 15 (11), 1119–1125.

15 Siedlecki, W. and Sklansky, J. (1989) A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10 (5), 335–347.

16 Bradley, P.S. and Mangasarian, O.L. (1998) Feature selection via concave minimization and support vector machines, in *International Conference on Machine Learning*, pp. 82–90.

17 Weninger, F. and Schuller, B. (2012) Optimization and parallelization of monaural source separation algorithms in the openBliSSART toolkit. *Journal of Signal Processing Systems*, 69 (3), 267–277.

18 Chen, W.Y., Song, Y., Bai, H., Lin, C.J., and Chang, E.Y. (2011) Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33 (3), 568–586.

19 Maschho, K.J. and Sorensen, D. (1996) A portable implementation of ARPACK for distributed memory parallel architectures, in *Proceedings of the Copper Mountain Conference on Iterative Methods, April*, pp. 9–13.

20 Lehoucq, R.B., Sorensen, D.C., and Yang, C. (1998) *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, SIAM.

21 Blei, D.M., Ng, A.Y., and Jordan, M.I. (2003) Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3, 993–1022.
22 Pritchard, J.K., Stephens, M., and Donnelly, P. (2000) Inference of population structure using multilocus genotype data. *Genetics*, 155 (2), 945–959.
23 Blei, D.M. (2012) Probabilistic topic models. *Communications of the ACM*, 55 (4), 77–84.
24 Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007) Object retrieval with large vocabularies and fast spatial matching, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8.
25 Dhillon, I.S., Fan, J., and Guan, Y. (2001) Efficient clustering of very large document collections. *Data Mining for Scientific and Engineering Applications*, 2, 357–381.
26 Schmitt, M., Janott, C., Pandit, V., Qian, K., Heiser, C., Hemmert, W., and Schuller, B. (2016) A Bag-of-audio-words approach for snore sounds' excitation localisation, in *Proceedings of the 14th ITG Conference on Speech Communication*, Paderborn, Germany, pp. 230–234.
27 Deng, J., Cummins, N., Han, J., Xu, X., Ren, Z., Pandit, V., Zhang, Z., and Schuller, B. (2016) The University of Passau Open Emotion Recognition System for the Multimodal Emotion Challenge, in *Proceedings of the 7th Chinese Conference on Pattern Recognition*, CCPR, Springer, Chengdu, P.R. China, pp. 652–666.
28 Liu, Z., Zhang, Y., Chang, E.Y., and Sun, M. (2011) Plda+: Parallel latent Dirichlet allocation with data placement and pipeline processing. *ACM Transactions on Intelligent Systems and Technology*, 2 (3), 26.
29 Wang, Y., Bai, H., Stanton, M., Chen, W.Y., and Chang, E.Y. (2009) PLDA: Parallel latent Dirichlet allocation for large-scale applications, in *International Conference on Algorithmic Applications in Management*, Springer, pp. 301–314.
30 Newman, D., Asuncion, A.U., Smyth, P., and Welling, M. (2007) Distributed inference for latent Dirichlet allocation, *Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS '07)*, Vancouver, British Columbia, Canada, pp. 1081–1088, Curran Associates Inc.,
31 Schmitt, M. and Schuller, B. (2017) openXBOW – Introducing the Passau Open-Source Crossmodal Bag-of-Words Toolkit. *Journal of Machine Learning Research*, 18, 3370–3374.
32 Schuller, B., Steidl, S., Batliner, A., Vinciarelli, A., Scherer, K., Ringeval, F., Chetouani, M., Weninger, F., Eyben, F., Marchi, E., Mortillaro, M., Salamin, H., Polychroniou, A., Valente, F., and Kim, S. (2013) The INTERSPEECH 2013 Computational Paralinguistics Challenge: Social Signals, Conflict, Emotion, Autism, in *Proceedings of the 14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, Lyon, France, pp. 148–152.
33 Schuller, B., Steidl, S., Batliner, A., Hantke, S., Hönig, F., Orozco-Arroyave, J.R., Nöth, E., Zhang, Y., and Weninger, F. (2015) The INTERSPEECH 2015 Computational Paralinguistics Challenge: Degree of Nativeness, Parkinson's & Eating Condition, in *Proceedings of the 16th Annual Conference of the International Speech Communication Association (INTERSPEECH), ISCA*, Dresden, Germany, pp. 478–482.
34 Schuller, B., Steidl, S., Batliner, A., Hirschberg, J., Burgoon, J.K., Baird, A., Elkins, A., Zhang, Y., Coutinho, E., and Evanini, K. (2016) The INTERSPEECH 2016 Computational Paralinguistics Challenge: Deception, Sincerity & Native Language, in

Proceedings of the 17th Annual Conference of the International Speech Communication Association (INTERSPEECH), ISCA, San Francisco, CA, pp. 2001–2005.

35 Schuller, B., Steidl, S., Batliner, A., Bergelson, E., Krajewski, J., Janott, C., Amatuni, A., Casillas, M., Seidl, A., Soderstrom, M., Warlaumont, A., Hidalgo, G., Schnieder, S., Heiser, C., Hohenhorst, W., Herzog, M., Schmitt, M., Qian, K., Zhang, Y., Trigeorgis, G., Tzirakis, P., and Zafeiriou, S. (2017) The INTERSPEECH 2017 Computational Paralinguistics Challenge: Addressee, Cold & Snoring, in *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, Stockholm, Sweden.

36 Zainuddin, Z. and Pauline, O. (2007) Function approximation using artificial neural networks, in *Proceedings of the 12th WSEAS International Conference on Applied Mathematics*, World Scientific and Engineering Academy and Society (WSEAS), MATH'07, pp. 140–145.

37 Abadi, M. et al. (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

38 Deng, J., Zhang, Z., Eyben, F., and Schuller, B. (2014) Autoencoder-based Unsupervised Domain Adaptation for Speech Emotion Recognition. *IEEE Signal Processing Letters*, 21 (9), 1068–1072.

39 Krizhevsky, A. and Hinton, G.E. (2011) Using very deep autoencoders for content-based image retrieval., in *ESANN 2011, 19th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 27–29.

40 Wang, N. and Yeung, D.Y. (2013) Learning a deep compact image representation for visual tracking, in *Advances in Neural Information Processing Systems*, pp. 809–817.

41 Socher, R., Huang, E.H., Pennington, J., Ng, A.Y., and Manning, C.D. (2011) Dynamic pooling and unfolding recursive autoencoders for paraphrase detection, in *Neural Information Processing Systems*, vol. 24, pp. 801–809.

42 Weninger, F., Bergmann, J., and Schuller, B. (2015) Introducing CURRENNT: the Munich Open-Source CUDA RecurREnt Neural Network Toolkit. *Journal of Machine Learning Research*, 16, 547–551.

43 Mousa, A.E.D. and Schuller, B. (2016) Deep bidirectional long short-term memory recurrent neural networks for grapheme-to-phoneme conversion utilizing complex many-to-many alignments, in *Proceedings of the 17th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, San Francisco, CA, pp. 2836–2840.

44 Hagerer, G., Pandit, V., Eyben, F., and Schuller, B. (2017) Enhancing LSTM RNN-based speech overlap detection by artificially mixed data, in *Proceedings of the AES 56th International Conference on Semantic Audio*, AES, Audio Engineering Society, Erlangen, Germany, pp. 1–8.

45 Liang, M. and Hu, X. (2015) Recurrent convolutional neural network for object recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3367–3375.

46 Pinheiro, P.H. and Collobert, R. (2014) Recurrent convolutional neural networks for scene labeling, in *International Conference on Machine Learning*, pp. 82–90.

47 Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015) Long-term recurrent convolutional networks for visual recognition and description, in *Proceedings of the IEEE Conference Computer Vision and Pattern Recognition (CVPR)*, pp. 2625–2634.

48 Keren, G. and Schuller, B. (2016) Convolutional RNN: an enhanced model for extracting features from sequential data, in *Proceedings of the International Joint Conference on Neural Networks (IJCNN) as part of the IEEE World Congress on Computational Intelligence (IEEE WCCI)*, Vancouver, Canada, pp. 3412–3419.

49 Maas, A.L., Le, Q.V., O'Neil, T.M., Vinyals, O., Nguyen, P., and Ng, A.Y. (2012) Recurrent neural networks for noise reduction in robust ASR, in *Proceedings of the 13th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, Portland, OR, pp. 22–25.

50 Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A.C., and Bengio, Y. (2015) A recurrent latent variable model for sequential data, in *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS '15), Volume 2, Montreal, Canada*, pp. 2980–2988, MIT Press, Cambridge, MA.

51 van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A. et al. (2016) Conditional image generation with pixelcnn decoders, in *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*, pp. 4790–4798, Curran Associates Inc.

52 van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016) Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*.

53 Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014) Generative adversarial nets, *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS '14)*, Volume 2, Montreal, Canada, pp. 2672–2680, MIT Press, Cambridge, MA.

54 Huang, G.B., Zhu, Q.Y., and Siew, C.K. (2006) Extreme learning machine: theory and applications. *Neurocomputing*, 70 (1), 489–501.

55 Lin, S., Liu, X., Fang, J., and Xu, Z. (2015) Is Extreme Learning Machine Feasible? A Theoretical Assessment (Part II). *IEEE Transactions on Neural Network Learning Systems*, 26 (1), 21–34.

56 Liu, X., Lin, S., Fang, J., and Xu, Z. (2015) Is an extreme learning machine feasible? A theoretical assessment (Part I). *IEEE Transactions on Neural Network Learning Systems*, 26 (1), 7–20.

57 Wang, L.P. and Wan, C.R. (2008) Comments on "The Extreme Learning Machine". *IEEE Transactions on Neural Networks*, 19 (8), 1494–1495.

58 Huang, G.B. (2008) Reply to 'Comments on "The Extreme Learning Machine"'. *IEEE Transactions on Neural Networks*, 19 (8), 1495–1496.

59 Chacko, B.P., Krishnan, V.V., Raju, G., and Anto, P.B. (2012) Handwritten character recognition using wavelet energy and extreme learning machine. *International Journal of Machine Learning and Cybernetics*, 3 (2), 149–161.

60 Qian, K., Janott, C., Pandit, V., Zhang, Z., Heiser, C., Hohenhorst, W., Herzog, M., Hemmert, W., and Schuller, B. (2016) Classification of the excitation location of snore sounds in the upper airway by acoustic multi-feature analysis. *IEEE Transactions on Biomedical Engineering*64 (8), 1731–1741.

61 He, K., Zhang, X., Ren, S., and Sun, J. (2016) Deep residual learning for image recognition, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

62 Agarwal, A., Akchurin, E., Basoglu, C., Chen, G., Cyphers, S., Droppo, J., Eversole, A., Guenter, B., Hillebrand, M., Hoens, T.R., Huang, X., Huang, Z., Ivanov, V., Kamenev, A., Kranen, P., Kuchaiev, O., Manousek, W., May, A., Mitra, B., Nano, O., Navarro, G., Orlov, A., Parthasarathi, H., Peng, B., Radmilac, M., Reznichenko, A., Seide, F., Seltzer, M.L., Slaney, M., Stolcke, A., Wang, H., Wang, Y., Yao, K., Yu, D., Zhang, Y., and Zweig, G. (2014) An Introduction to Computational Networks and the Computational Network Toolkit. *Microsoft Technical Report MSR-TR-2014-112.*

63 Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012) Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590.*

64 Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011) Torch7: A Matlab-like environment for machine learning, in *BigLearn*, NIPS Workshop, EPFL-CONF-192376.

65 Chollet, F. et al. (2015) Keras, https://github.com/fchollet/keras.

66 Amiriparian, S., Gerczuk, M., Ottl, S., Cummins, N., Freitag, M., Pugachevskiy, S., and Schuller, B. (2017) Snore sound classification using image-based deep spectrum features, in *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, Stockholm, Sweden.

67 Walt, S.V.D., Colbert, S.C., and Varoquaux, G. (2011) The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13 (2), 22–30.

68 Hunter, J.D. (2007) Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9 (3), 90–95.

69 Freitag, M., Amiriparian, S., Gerczuk, M., Cummins, N., and Schuller, B. (2017) An 'End-to-Evolution' Hybrid Approach for Snore Sound Classification, in *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, ISCA, Stockholm, SE.