Preprint from Braschler, Stadelmann, Stockinger (Eds.): *“Applied Data Science - Lessons Learned for the Data-Driven Business”*, Springer , 2018 (to appear) .

Beyond ImageNet - Deep Learning in Industrial Practice

Thilo Stadelmann, Vasily Tolkachev, Beate Sick, Jan Stampfli, and Oliver Dürr.

Abstract *Deep learning (DL) methods have gained considerable attention since 2014. In this chapter we briefly review the state of the art in DL and then give several examples of applications from diverse areas of application. We will focus on convolutional neural networks (CNNs), which have since the seminal work of Krizhevsky et al. (2012) revolutionized image classification and even started surpassing human performance on some benchmark data sets (Ciresan et al., 2012a, He et al., 2015a). While deep neural networks have become popular primarily for image classification tasks, they can also be successfully applied to other areas and problems with some local structure in the data. We will first present a classical application of CNNs on image-like data, in particular, phenotype classification of cells based on their morphology, and then extend the task to clustering voices based on their spectrograms. Next, we will describe DL applications to semantic segmentation of newspaper pages into their corresponding articles based on clues in the pixels, and outlier detection in a predictive maintenance setting. We conclude by giving advice on how to work with DL having limited resources (e.g., training data).*

1. Introduction to deep learning

Deep neural networks have been greatly influencing the world of pattern recognition for several decades (Schmidhuber, 2014). The disruptive nature of the approach became obvious to a wider audience since Krizhevsky et al. (2012)’s exploit on the ImageNet task. Since then the corresponding gain in perceptual performance has often been such that error rates could be halved or even improved by an order of magnitude with respect to the previous state of the art on open benchmark datasets (LeCun et al., 2015). In this chapter, we show how deep learning (DL) methods can be applied not only to classical computer visions tasks from research, but to a wide variety of tasks in industry beyond classification.

While it is easy for humans to recognize someone’s speech or classify objects in an image, problems like these had previously posed a serious challenge for computers for a long time. In the traditional pattern recognition paradigm, researchers tended to manually design informative features, on which classification algorithms were applied. In computer vision, these were, among others, Haar features or Gabor filters (Szeliski, 2010). In speech recognition, one used, for example, Mel frequency cepstrum coefficients (Zheng et al., 2001), while in NLP, there were n-gram features or mutual information between the words (Bouma, 2009). These features were burdensome to engineer manually and it was unclear which ones were the most informative for the task at hand.

DL revolutionized the field by offering end-to-end learning, starting at almost raw data input without the need for kernel or feature engineering and allowing a hierarchy of neural network layers to learn the necessary features on its own. In the following paragraphs we provide a brief overview of these developments.

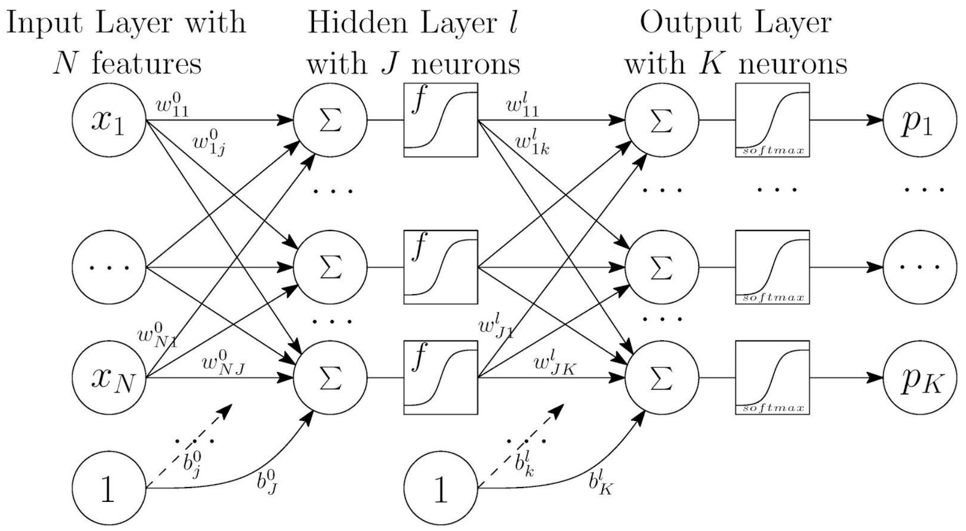

Figure 1. A feed-forward neural network with features $x_1, \ldots, x_N$ and label probabilities $p_1, \ldots, p_K$.

1.1 Fully connected neural networks for classification

The simplest architecture, from which the development in the field of neural networks started, is a fully connected feed-forward neural network (Rosenblatt, 1957). It can be considered as a directed acyclic graph where the information flows from left to right (see Figure 1). A neuron is made up of a circle (summing up the inputs), followed by a square (depicting a nonlinear activation function that serves as a threshold). Inspired by a biological brain, each neuron $z^l_j(x)$ in a layer of neurons (vector[1] $z^l(x)$) receives an input from all the neurons from the previous layer z^{l-1} with a weight matrix W. The weighted sum of inputs for the neuron is then passed through a nonlinear activation function f inside the neuron that acts as a trainable threshold: if f receives a high value, the neuron is activated and passes the transformed signal to the neurons in the next layer on the right. In general, the output of all neurons of a layer l can be recursively described with the weight matrices $W^{l-1}, W^{l-2}, \ldots$ and bias vectors $b^{l-1}, b^{l-2}, \ldots$ as:

$$z^l(x) = f({w_0}^{l-1} + W^{l-1} z^{\,l-1}(x)) = f({w_0}^{l-1} + W^{l-1} f({w_0}^{l-2} + W^{l-2} z^{\,l-2}(x))) = \ldots$$

The network can possess many hidden, interconnected layers. For classification into classes $1, 2, \ldots, K$, its last (output) layer will have as many nodes as there are classes to distinguish in the data (K in this case). To obtain probabilities $P(y_k | x)$ for each class k, the raw aggregated inputs (scores) must be standardized by their sum over all classes to produce values between 0 and 1, which is usually done with the softmax function (Bishop, 2006, p.115):

$$P(y_k | x) = \frac{exp(-\sum_{j=1}^{J} {w_{jk}}^l {z_i}^l(x))}{\sum_k exp(-\sum_{j=1}^{J} {w_{jk}}^l {z_i}^l(x))}$$

where ${w_{jk}}^l$ are the learned weights of layer l which are elements of the matrix W^l. Historically, sigmoid $f(x) = (1 + e^{-x})^{-1}$ and hyperbolic tangent $f(x) = tanh(x)$ were used as activation functions; now it is recommended to use a Rectified Linear Unit (ReLU) $f(x) = max(0, x)$ which significantly speeds up training because of improved gradient flow (Krizhevsky et al., 2012).

[1] Vector arrows are usually not drawn in the DL literature.

To train the neural network (i.e., find the optimal weights), a loss (discrepancy between the true and predicted classes) is computed once the signal is propagated to the last layer and the class probabilities are calculated. Then the weights are adjusted to minimize the loss function, which is usually done with a maximum likelihood approach: The weights are optimized by stochastic gradient descent (SGD) (Goodfellow et al., 2016) and the required gradients are efficiently calculated making use of the chain rule. For example, to calculate the gradient of the loss at the layer $l-2$ one can use the gradient at the layer $l-1$:

$$\frac{\partial Loss}{\partial z^{l-2}} = \frac{\partial Loss}{\partial f}\frac{\partial f}{\partial z^{l-1}}\frac{\partial z^{l-1}}{\partial z^{l-2}}$$

The gradient thus propagates 'back' from the loss to previous layers. Therefore, this procedure is also called backpropagation in the context of neural networks (Rumelhart et al., 1988); a gentle introduction is provided by Nielsen (2015). Training usually runs on GPUs for computational reasons (speed-up of an order of magnitude as compared to CPUs), and the training data is split into so-called mini-batches, which fit into the GPU memory and on which SGD is run. Nowadays, more advanced variants of SGD like ADAM (Kingma & Ba, 2014) and ADADELTA (Zeiler, 2012) are usually employed instead of the standard SGD and should be preferred.

1.2 Convolutional Neural Networks (CNNs)

While fully connected networks possess significant flexibility, they have many parameters and tend to significantly overfit the data while not capturing any local structures such as the 2D correlations of pixels in images. CNNs were introduced in order to resolve these issues.

The first neural networks with convolutional filters date back to the work of Fukushima (1980). They were made trainable end-to-end via backpropagation by LeCun et al. (LeCun et al., 1998a), yet their applicability was limited at that time due to scarce computing capacities and the shortage of large labeled datasets. The revival came in 2012 with the works of Ciresan et al. (2012a) and Krizhevsky et al. (2012), who independently presented significantly deeper nets trained on modern graphics cards (GPUs). This GPU training enabled increased depth by exploiting the cheap hardware originally developed for 3D games, using its massively parallel matrix computation capabilities. It was thus possible to solve the problems of the traditional approach and completely outperform it in numerous pattern recognition challenges. Currently, deep CNNs have error rates as low as humans (or sometimes even better (Nielsen, 2017)) in many tasks, including image classification (He et al., 2015a), geolocation (Weyand et al., 2016), speech recognition (Xiong et al., 2016), lip reading (Chung et al., 2016), as well as the games of GO (Silver et al., 2016) and poker (Moravcik et al., 2017).

The intuition behind CNNs goes back to the physiological experiments of Hubel and Wiesel (1959) on the response of visual neurons in a cat's brain to various oriented stimuli. The main idea of CNNs is to design a neural network that can easily exploit local structure in its input in hierarchically organized layers to extract subsequently more abstract features: convolutional kernels (resembling the filters of classical image processing[2]) are slid over the

[2] In digital image processing, to apply a filter (or kernel) to a specific region of an image, centered around a specific pixel, means to take the weighted sum of pixels in the center pixel's neighborhood. The size of the neighborhood is determined by the filter size (e.g., *3x3* pixels), whereas the weights are determined by the filter designer. Numerous classical filters for all kinds of image processing tasks are known. For example, to smoothen an image, one applies a filter with each of the *N* weights equaling *1/N*, so the filter response is an average over the filter's spatial area. The filters "filter 1" and "filter 2" in Figure 2. show vertical and horizontal edge detectors, respectively (when white pixels stand for a weight of *-1* and blue pixels for a weight of *1*, or vice versa). In CNNs, the filter weights are

complete input and the dot product of the input with the kernel at each location is computed. Thereby, each possible kernel location shares the same weights, which massively saves parameters in this layer, which in turn can be "invested" back into additional layers. Convolutional layers usually alternate with some sort of sub sampling (originally: pooling) layers and thus allow the CNN to abstract local structure to global insights.

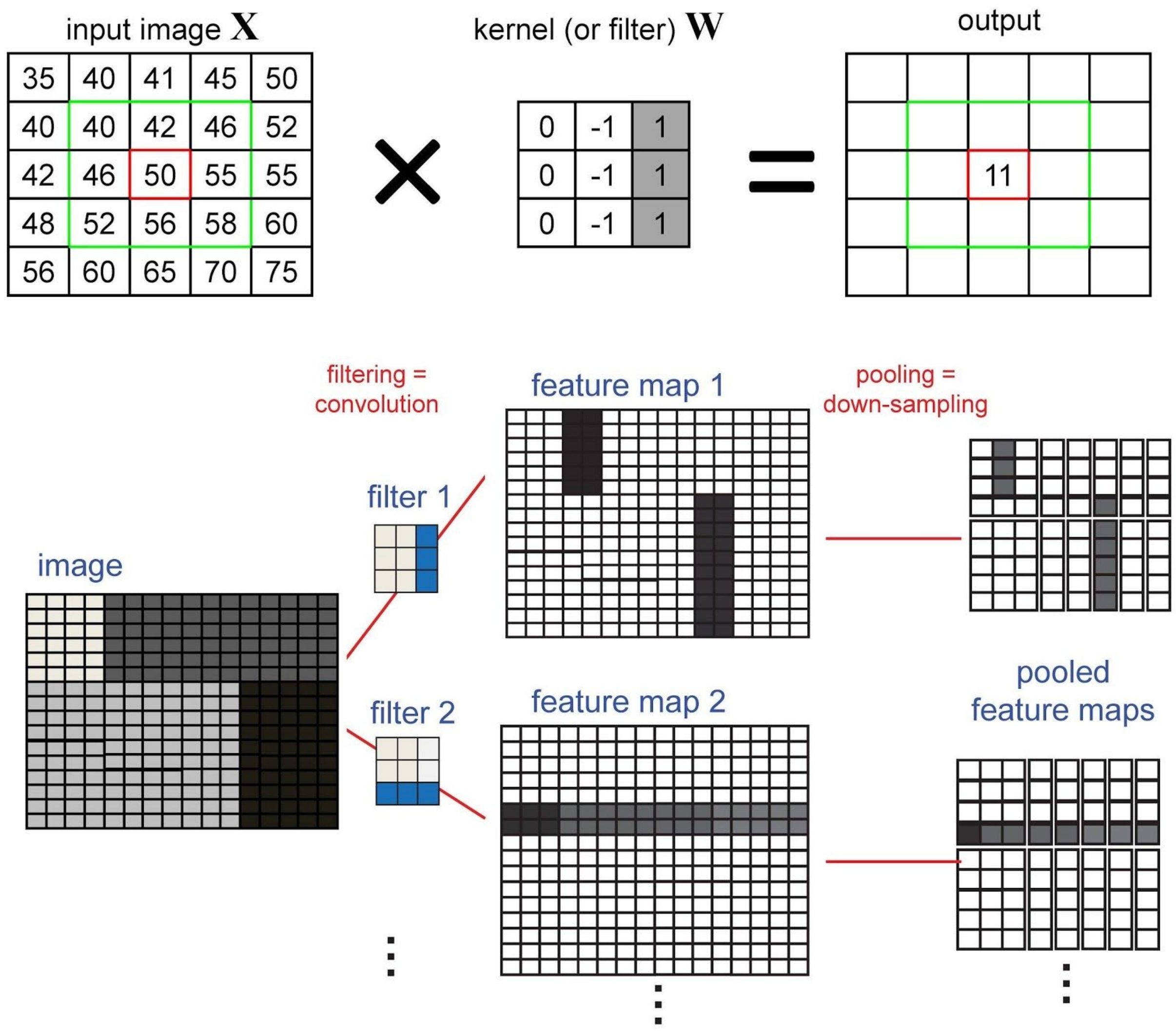

Figure 2. Convolutional filters (top) slide over the images, creating feature maps, which are then down-sampled to aggregate the most important information about the image structure.

The convolutional operation makes sense because it processes information locally and converts it to a feature map (which is the output of a specific filter, evaluated at every pixel of the current layer, resembling the filtered image) that indicates the presence or absence of the very feature the convolutional filter describes using its learned coefficients. A learned feature could be, for example, an edge, a simple shape, or a combination of these in the later layers. The feature map can then be compressed by means of a down-sampling

learned, while the size and number of filters are chosen hyperparameters. This means that each convolutional layer in a CNN can learn any classical image transformation (see https://en.wikipedia.org/wiki/Digital_image_processing), one per filter (you see the number of filters by counting the number of feature maps in the next layer, cp. Figure 2).

operation (e.g., max pooling[3]) to create a global big picture of the input contents out of the local features (see Figure 2). In CNNs, several blocks of convolution and down-sampling are thus stacked in the network with various input sizes to achieve sufficient generality and to capture enough detail, so that every block is responsible for some image property. As a result, a hierarchical representation of object properties is built by the convolutional layers. Finally, a fully-connected output layer produces class probabilities.

To cope with varying input image sizes and to produce richer outputs than just class labels (e.g., full images again), the fully convolutional network (FCN) has been proposed (Long et al., 2014), which implements all layers (also down-sampling and fully connected ones) using convolutions only (see Section 4).

Overall, a CNN still contains a lot more free/trainable parameters (usually in the order of hundreds of millions) than observations used for training, so that with a "wrong" training procedure it is easy to overfit the data. There are a number of possible solutions which are now application standards. First, traditional CNNs are not intrinsically invariant to transformations like object rotations, flips, lighting etc. In order to enrich the training data, it is common to do image augmentation prior to training that reflects the input's nature (i.e. apply transformations like rotation, translation and random scaling to the data and adding natural "noise", using the transformed images for training as well). Second, a regularization technique called dropout (Srivastava et al., 2014) was introduced to significantly reduce overfitting, which consists of randomly deactivating each neuron in a layer usually with a probability of 0.5 at training.

Wrong weight initialization in a network can pose a serious problem as well. With an inappropriate initialization, some of the neurons may soon come into an over- or under-saturated regime, and, depending on the activation function, the gradients will be close to zero. This in turn means that there would be almost no update of the weights during backpropagation, and parts of the network will die out. To avoid this and other problems, including overfitting, the batch normalization technique (batchnorm) has been proposed (Ioffe & Szegedy, 2015). It consists of standardizing a mini-batch *at each layer* of the network with its mean and standard deviation after each training iteration in order to keep a stable input distribution to each neuron, thus facilitating gradient flow. Moreover, batchnorm also allows to learn the shift and scale normalization parameters to undo the standardization when needed. Batchnorm alleviates the dependence on initialization, allows faster learning rates and training times and acts as a regularizer due to a more even sampling in the parameter space.

To summarize, the use of GPUs in conjunction with the abundance of large (annotated) datasets made CNNs applicable to a wide range of problems. This was only possible in combination with the algorithmic improvements outlined earlier (i.e., ReLU activation, batchnorm initialization of the weights, ADAM or ADADELTA optimizer, dropout

[3] Max-pooling describes the process of moving a kernel of e.g. 2x2 pixels over an image-like representation (a layer in the neural network); for each location, only the maximum pixel value is carried over to the next layer, thus resulting in down-sampling the original 2x2 pixels (to keep the example from above) information to just 1x1. The size of the kernel as well as its step size (stride) typically are hyperparameters of the neural network architecture. However, in some modern designs, the architecture offers down-sampling at various degrees after each convolutional step, with the possibility to learn during training for the task at hand which of several alternative paths through the network should be followed at which layer. Thus, it offers to "learn" the degree of downsampling to a certain extent (Szegedy et al., 2014)(He et al, 2015b).

regularization, and data augmentation) – compare (Szegedy et al., 2014). All these improvements are now implemented in modern software frameworks used for production-ready deep learning, such as TensorFlow[4] or Torch[5], or included in high-level libraries on top of these frameworks like Keras[6] or TFLearn[7]. These frameworks also offer a collection of pre-trained networks available for many image recognition tasks. They can be adapted to similar tasks using transfer learning (Pan and Yang, 2010), eliminating the need for time-consuming training from scratch, which could still take 1-2 weeks for any real-world task even on modern hardware.

1.3 Non-obvious use cases

In the following sections, we describe various applications of deep neural networks. We focus on non-classical tasks, given that the performance of CNNs on image classification tasks is well known. Table 1 gives an overview of the selected tasks with a focus on the properties of every use case. Moreover, the table describes in which section of this chapter the use case is described in more detail. Table 2 summarizes the special challenge of each task and the main deviation from the classical image classification approach.

Sec.	Application	Type of final task	Training data	Results
2	Cell phenotype classification for drug discovery	classification (supervised)	ca. 40k images having 5 color channels	outperforms state of the art (better than LDA and SVM)
3	Media segmentation according to voice	clustering (unsupervised)	spectrograms of raw audio (ca. 25s on average for each of 100 speakers)	outperforms state of the art (better than hand-coded features and statistical models)
4	Newspaper segmentation into articles	semantic segmentation (supervised)	ca. 430 scans of newspaper pages (+ additional input from OCR) + 5k partially labeled pages (+OCR)	outperforms state of the art (better than classification CNN)
5	Predictive maintenance of rotating machinery	anomaly / outlier detection (unsupervised)	spectrograms of ca. 1k raw vibration signal measurements	on par with state of the art (SVM, PCA, statistical models)

Table 1: Overview of task properties for each of the following use cases.

Sec.	Non-obvious because?	Solved by?
2	introductory case, but 5 color channels instead of the usual 1 to 3	straightforward extension of standard model using data augmentation on training data
3	audio instead of image as input; final goal is a clustering	input is converted to a spectrogram to be treated as an image; output is a learnt representation to be clustered offline by another method
4	output is a cutting mask (outline of the text columns & images that make up an article on a page)	output is an image of the same size as the input: pixels of same color indicate areas belonging to the same article
5	training data has only one class, model shall	using an autoencoder architecture for the network

[4] https://www.tensorflow.org/
[5] http://torch.ch/
[6] https://keras.io/
[7] http://tflearn.org/

	indicate if new data deviates from it (instead of segregating it from a well-specified 2nd class)	and interpreting the reconstruction error as the degree of novelty in the test signal

Table 2: What makes the following tasks special and how can this be handled using deep learning?

We start with an application of CNNs to fluorescence microscopy images, an application which up to now requires much tedious and time consuming work from highly trained experts in biology and image analysis. We then continue to speaker clustering, where pre-trained CNNs are used to extract learned feature vectors per speech utterance for subsequent hierarchical clustering. This is followed by an application in which a fully convolutional network segments the pixels of a scanned newspaper page into sets of semantically belonging articles. Finally, the use of DL for predictive maintenance is illustrated before we conclude by giving an outlook on how to generally apply deep nets in contexts with usually very limited training data and computational resources.

2. Learning to classify: Single cell phenotype classification using CNNs

High content screening (HCS) is an essential part of the drug discovery pipeline used in the pharmaceutical industry. Screening involves the application of many thousands of drug candidates (compounds) to living cells with the aim to investigate the cell response. This response manifests itself in the change of the phenotype. Examples for those phenotypes are: dead cells, dying cells (apoptosis), dividing cells (mitosis), and cells expressing certain proteins.

In simple settings an applicable approach for classification is to extract predefined features for each cell (e.g. diameter, area, circumference of the nucleus or the cell, the intensity of different fluorescent stains in the nucleus or the cell or other organelles) and use them as an input for classification (see Figure 3, upper right panel). Such pre-defined features can be extracted by a specialized software such as CellProfiler[8]. However, more challenging cases require a tailored image analysis procedure to extract appropriate features, which needs to be done from scratch for each experiment, requiring both in-depth knowledge of cell biology and advanced knowledge of image processing.

Deep learning, on the other hand, does not rely on those predefined or hand-crafted features, and employs only labeled data, a task which is feasible for a biologist without a profound competence in image processing. Hence, deep learning has the potential to radically change the workflow in HCS. The envisioned CNN approach allows to learn the features and the classification model in one training procedure (see Figure 3, lower right panel).

2.1 Baseline approach

We use part of the image set BBBC022v1 (Gustafsdottir et al., 2013) (the "Cell Painting" assay), available from the Broad Bioimage Benchmark Collection (Ljosa et al., 2012). We analyze the images of human cells treated with 75 compounds – each compound resulting in one of 3 phenotypes (named A,B,C). In addition, we add the phenotype D of the cell without treatment (mock class). In total, we have the following number of detected cells, which were

[8] http://cellprofiler.org/

imaged in 21 different wells on 18 different plates: 40,783 (mock class), 1,988 (cluster A), 9,765 (cluster B), and 414 (cluster C).

Approximately 20% of the data is put aside for testing, containing the following number of examples per class: 8,217 (mock class), 403 (cluster A), 1,888 (cluster B), and 82 (cluster C) cells from 10 different wells on 5 different plates. The remaining 80% of the data is used to train and tune different classifiers comprising a CNN based on the raw image data as well as the three baseline approaches often used in HCS (Dürr et al., 2007): Fisher linear discriminant analysis (LDA), Random Forest (RF), and support vector machine (SVM), based on CellProfiler features. Before the input into CNNs, 5 images of size 72x72 are cropped for each cell from the original images. The bounding box is constructed to be quadratic so that the entire cell is within the box.

For the baseline workflows, each of the extracted features is normalized to have zero mean by a z-transformation. We then use the following implementations and parameterizations for classification: a SVM with a linear kernel (the penalty parameter C of the SVM is optimized using a 10 fold cross-validation on the training set); a RF with the default value of 500 trees; and LDA. All algorithms have been implemented in Python using the scikit-learn library[9].

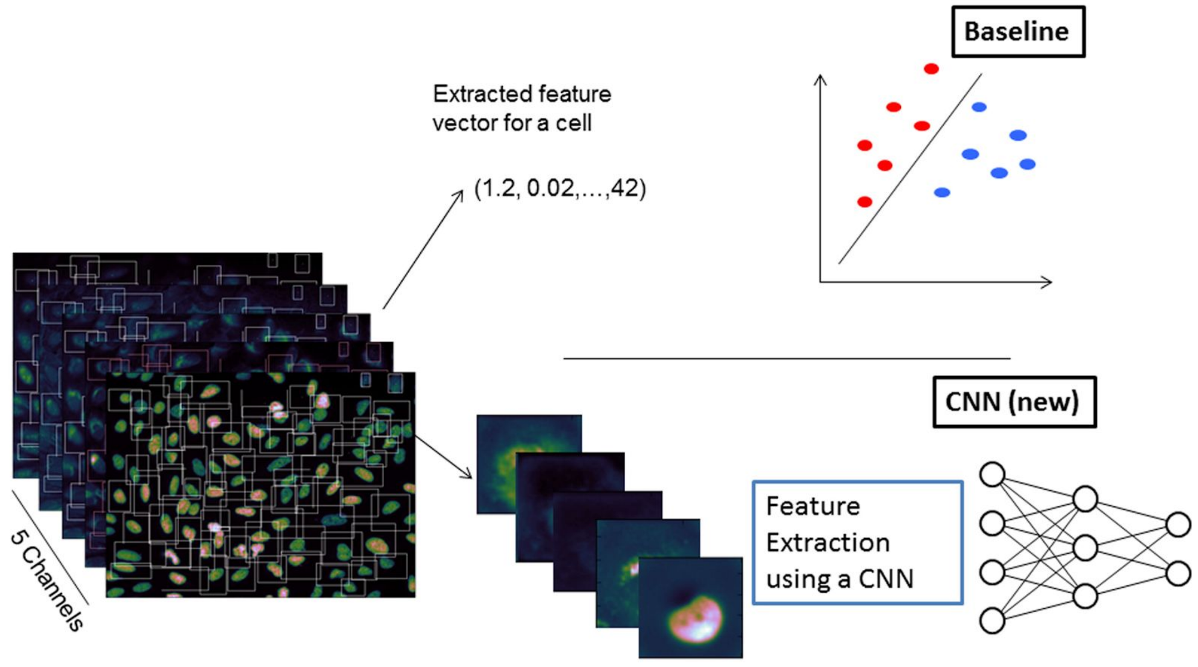
Baseline
Extracted feature vector for a cell
(1.2, 0.02,…,42)
5 Channels
CNN (new)
Feature Extraction using a CNN

Figure 3. Overview of the used analysis scheme. The baseline approach (upper part) needs handcrafted features, which are extracted using CellProfiler prior to classification using e.g.the SVM. In the CNN approach (lower part), the features are learned automatically.

2.2 CNN analysis

As the only preprocessing step for CNN, we normalize the values per pixel. The architecture of the CNN is inspired by the second-best entry of the 2014 ImageNet competition (Simonyan & Zisserman, 2015). All convolutional filters (C) have the size of (3,3) and a stride of 1 pixel and use ReLU activations; no padding is applied at the boundaries. Two convolutional layers are followed by a (2,2) max-pooling layer, forming a stack. Our network consists of 3 such stacks, which have 32, 64, and 128 kernels each. These stacks are followed by 3 fully connected layers with 200, 200, and 50 nodes respectively, and a final softmax layer for the 4 classes. The network has about 1.2 million learnable weights. For

[9] http://scikit-learn.org/stable/

learning the weights of the network, we split the data available for training into two parts: one part is used for fitting the weights (training set), the other 20% are used for validation (validation set). Note that the test set described above is only used for the evaluation of the trained CNN.

To prevent overfitting, we use dropout for the hidden layers, setting a fraction of $p = 0.3$ of all nodes randomly to zero in the training phase. We further used data augmentation to artificially enlarge the training set by applying the following random transformations on each image after each epoch (one epoch comprises a full pass through the training set): a random rotation uniformly chosen in the range of 0 to 360 degrees; a random translation up to 5 pixels in each direction (uniformly chosen); and a scaling with a scaling factor uniformly chosen in the range 0.9 to 1.1.

The network is implemented using the nolearn extension of the Lasagne python library[10]. All runs have been done on an off-the-shelf PC with a NVIDIA GeForce GPU.

2.3 Results and discussion

The training of the CNN took on average 135 seconds per epoch when using augmentation of the training data; without augmentation an epoch took just 70 seconds. The network was trained for 512 epochs (18 hours). Without augmentation, we were overfitting already after about 20 epochs, meaning the training loss continued to decrease, but the validation loss on the validation set (which was not used for parameter optimization) began to deteriorate. When using the data augmentation strategy as described above we avoided overfitting even after 512 epochs. Averaged over the last 100 epochs, the validation accuracy is (0.9313 mean, 0.0079 std).

We applied the learned network to the test set consisting of 10590 cell images. In contrast to the long training phase, the prediction of the probabilities for the 4 classes only takes approximately 6.9 seconds for all images. The overall accuracy on the test set is 93.4%. The confusion matrix is shown in Table 2 together with the best baseline approach (LDA).

	DMSO (True)	Cluster A (True)	Cluster B (True)	Cluster C (True)
CNN				
DMSO	7775	13	208	0
Cluster A	28	382	23	1
Cluster B	414	8	1657	0
Cluster C	0	0	0	81
LDA				
DMSO	7949	20	542	0
Cluster A	15	323	35	12
Cluster B	251	60	1310	1
Cluster C	2	0	1	69

Table 2: Results of baseline and CNN approach on the test set.

In this HCS study the CNN trained with raw images yields the best classification accuracy when compared to three state-of-the-art image analysis approaches with the traditional pipeline of image feature extraction followed by training a classifier based on those features. Besides the better performance of the CNN-based approach, it has additional benefits such as saving time and costs during the image analysis step and providing high robustness and broad application range.

[10] http://lasagne.readthedocs.io

3. Learning to cluster: Extracting relevant features for speaker diarization

Speaker diarization is the task of segmenting an audio recording of a meeting, a lecture, a political debate or some broadcast media by speaker identity to answer the question *"who spoke when"* (Beigi, 2011). No prior knowledge about the number or specific identities of participating speakers is assumed. If we assume a pre-segmentation into speaker-specific segments by some other process, but still need to answer the question of which segments belong to the same speaker and how many speakers exist, the task is called *speaker clustering*. Typical business use cases arise as a preprocessing step to general media indexing (in order to make it searchable), specifically in media monitoring (e.g., who has been covered on radio), meeting summarization (e.g., to search by panelist) or the evaluation of qualitative interviews in psychological research.

Speaker clustering is typically approached by first extracting base audio features like Mel-frequency cepstrum coefficients (MFCC) for the whole audio stream (Ganchev et al., 2005), followed by a segment-wise modeling (e.g., using adapted Gaussian mixture models (Reynolds et al., 2000)) to create higher-level speaker-specific features per segment (e.g., i-vectors (Dehak et al., 2011)). These higher-level features of each segment are then subject to a clustering process. Typically, agglomerative hierarchical clustering is used (Kotti et al., 2008).

In general, clustering is viewed as the prototypical example of an unsupervised learning task, using algorithms like k-means (MacQueen, 1967) or DBSCAN (Ester et al., 1996) as alternatives to hierarchical clustering. As with supervised learning schemes, these algorithms have their inductive biases (Mitchell, 1980). They will find structure in the data if and only if (a) that structure is reflected in the extracted features, and (b) the structure fits what the algorithm is biased to look for. K-means, for example, will find structure expressed in the mutual distances between data points and hypothesized cluster centers, while DBSCAN finds clusters only if they are reflected in the density structure of the data set.

In general, clustering is also close in spirit to the task of classification: while a classifier groups the test data into any of a pre-defined number of classes, a clustering algorithm basically has the same goal of grouping test data together – just that the number and identity of classes/clusters is not predefined. Given the success of deep neural networks in classification on the one hand, and their general ability to extract meaningful and task-specific features from almost raw data on the other hand (Razavian et al., 2014), it seems compelling to bring these properties to bear on the task of speaker clustering.

3.1 Supervised learning for improved unsupervised speaker clustering

The typical deep learning approach to clustering uses the neural network as a data-driven feature extractor to transform the input into so-called embeddings (Mikolov et al., 2013) (Romanov & Rumshisky, 2017). Each embedding is then used as the new representation of the input vector and fed into a subsequent clustering process using one of the abovementioned classic algorithms. The embedding is found for a respective input by extracting the activations of one of the upper layers of the neural network, which has previously been trained for a related or "surrogate" task.

For this setup to be successful for speaker clustering, it is important that the learned embeddings (or high-level features) incorporate the following ideas:

- *Contain prosodic information:* Stadelmann and Freisleben (2009) highlighted the importance of the evolution of a sound using short segments of ca. 120 ms in length for human-level recognition performance (i.e., temporal information matters instead of a pure bag-of-frames approach (Aucouturier et al., 2007)).
- *Be voice-specific:* When the surrogate task to train the feature-extracting network is speaker identification using a discriminative model, chances are that the extracted features are better suited to distinguish the specific set of speakers used during training from each other (rather than modeling what makes any voice unique, which is what is needed for clustering).

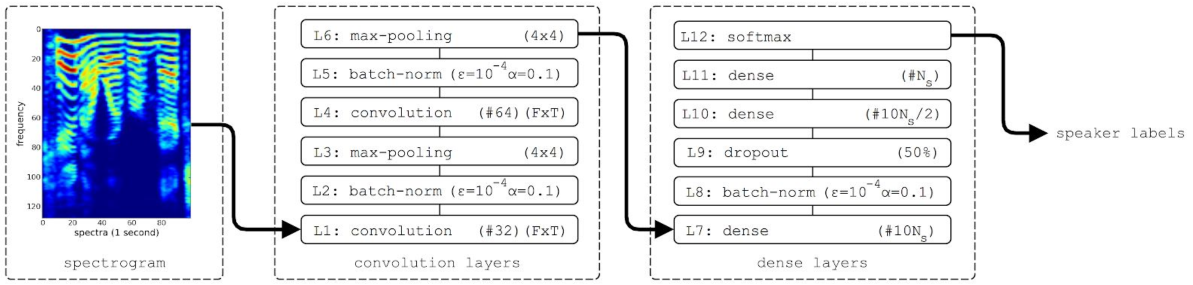

Figure 4: Architecture of the CNN used to extract speaker embeddings (Lukic et al., 2017).

We use spectrograms[11] as input and built up a CNN architecture inspired by Dielemann and Schrauwen (2014) to extract embeddings based on these two principles, and evaluated it on the well-known TIMIT speech corpus. The architecture is shown in Figure 4; Lukic et al. (2016-2017) give all details. The rationale behind this setup is twofold: First, we address the temporal aspect mentioned above ("*prosodic information*") by using convolutional networks: the convolutional layers are able to extract time-dependent aspects of a voice through 2D convolutional kernels that operate on the spectrograms and thus operate on the time axis. Second, the loss function of Hsu and Kira (2015) ensures that the embeddings explicitly focus on being similar for identical speakers ("be voice-specific"), and dissimilar for different speakers (irrespective of the concrete speaker identity). This ensures a proper closeness of the surrogate supervised training task to the final task of clustering (i.e., grouping voices by closeness of their embeddings).

3.2 Results

We took the first n speakers in lexicographic ordering from the TIMIT test set for the clustering experiment. We divided the 10 sentences per speaker into two utterances by taking the first 8 sentences (lexicographically ordered by filename) for utterance one, and the last two for the second utterance. Utterance one is approximately 20 seconds long on average, while utterance two is ca. 5 seconds long. Using the architecture and experimental setup described in greater detail in (Lukic et al., 2017), we have been able to cluster up to $n = 80$ speakers with a reasonable misclassification (MR) rate of $13.75\,\%$[12]. The best reported previous results worked only for up to 40 speakers with an MR of $5\,\%$, which is on par with our approach. Ca. 14% MR are a starting point for unsupervised media indexing

[11] A spectrogram is a 2D image representing a time-frequency decomposition of an audio signal: the x-axis represents time, the y-axis represents frequency, and the color encodes energy (compare the leftmost part of Figure 4, showing 3 seconds of voiced speech).

[12] MR counts the share of utterances that are grouped into the wrong cluster. Wrong can mean two things: utterances of different speakers are either joined into the same cluster, or utterances of one speaker are distributed over several (pure) clusters instead of combined to a single one.

tasks, but should be improved in the future. The main message in this result is that now automatic indexing becomes feasible because it can cope with practically relevant speaker set sizes.

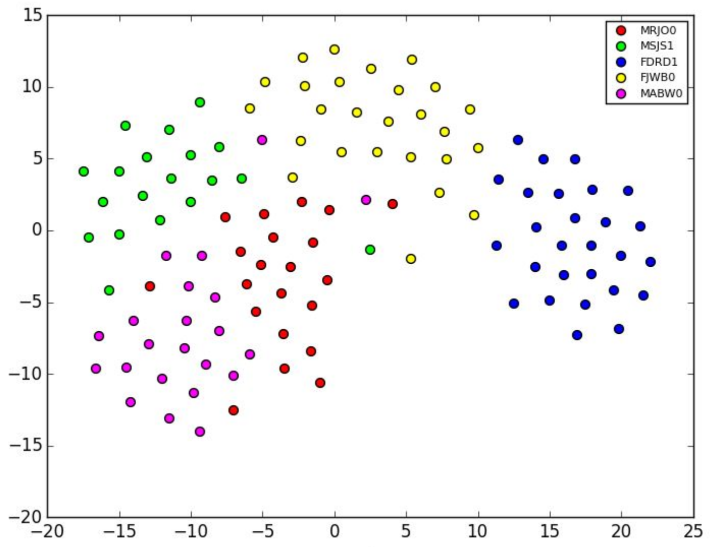

Figure 5: A t-SNE visualization of the embeddings of several speech segments from 5 TIMIT speakers.

Figure 5 allows for a qualitative assessment of the embeddings of $n = 5$ speakers. The used t-SNE visualization method performs nonlinear dimension reduction from the dimensionality of the embedding vectors into 2D (van der Maaten & Hinton, 2008) while preserving the original similarity structure as much as possible. We observe that overall the embeddings of any speaker group together nicely.

We conclude that for the task of speaker clustering, the sequence-learning capabilities of the CNN architecture together with the Kullback-Leibler divergence-related loss function enable the extraction of voice-specific features for subsequent clustering. The involved learning task seems to be quite non-trivial: only by using batchnorm and 30,000 epochs of training using ADADELTA (Zeiler, 2012) we were able to produce useful results. A next step would be to embed the clustering in a truly end-to-end optimizable process that includes the actual clustering.

4. Learning to segment: FCNs for semantic segmentation of newspaper pages

Newspapers are provided and consumed to a large extent in printed form. Large archives of such papers do exist, containing a historically important cultural heritage. In order to analyze or search them, they need to be available in a suitable digital form. Since newspapers consist of articles that can be considered as independent units, one usually wants to access these semantically meaningful units directly instead of whole pages. Therefore, digitization of newspapers not only needs optical character recognition (OCR) (Mori et al., 1999), but also semantic segmentation (Long et al., 2014). The term semantic segmentation means to "cut" a page into connected components (headers, text, images) that together constitute a semantic unit we call an article. In the use case of media monitoring, today's products and services are very costly because this segmentation work has to be done manually. This also means that no real-time monitoring is possible, and neither is the processing of larger archives feasible using manual work.

In this case study, we improve a straightforward application of a classification CNN by a much better suited network architecture to achieve practically useful segmentation results of newspaper pages into sets of semantically connected articles. Both approaches are based on CNN architectures and provide segmentation masks which can be used to extract the articles from the corresponding newspaper pages. A segmentation mask is a binary image with black pixels standing for articles and white pixels for borders. In order to extract articles using the segmentation mask, we apply a post-processing step to get the coordinates of the (black) article areas matching the original scans.

Our dataset consists of 507 high resolution scans (i.e., images) of newspaper pages from the papers with highest circulation among Swiss newspapers, ranging from classical weekly newspapers to boulevard media. It is accompanied by manually created segmentation masks as ground truth. We transform the original scans of the newspaper pages to simplified representations to be used as input for our CNNs. This is done by replacing illustrations with gray areas and by blackening lines of texts (after OCR). In the end, our data set contains 507 pictures with two channels each (plus ground truth segmentation mask, see Figure 6): the original scan, and the abovementioned transformation. We use approximately 85% of the dataset for training and hold out the remaining 15% for testing. In addition to this fully labelled dataset, we have ca. 5'500 partially labelled pages (i.e., each page contains also unsegmented articles).

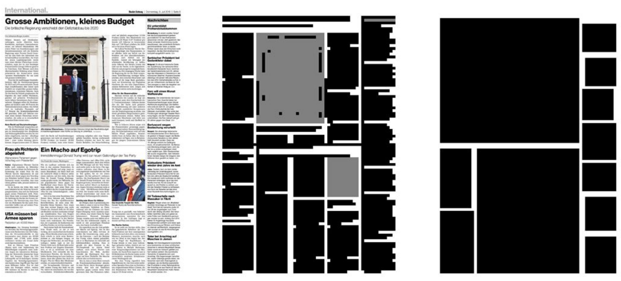

Figure 6. Example of our dataset showing an original scan of a newspaper page (left), the transformed representation (middle) and the manually created segmentation mask (right).

4.1 CNN-based pixel classification vs. one-pass FCNs

A straightforward first approach is based on the work of Ciresan et al. (2012b): we use a CNN-based pixel classification network (PCN) to classify a newspaper page pixel by pixel using subsequent applications of the CNN to every pixel. The class of each pixel (article or border) is predicted from pixel values in a 25x25 pixels square window centered on it. The network is trained by using only the transformed images, for which we adjust the resolution so that all of them have a height of 100 pixels without changing the aspect ratio. Classifying such an image with e.g. 100x75 pixels results in 7,500 windows. We therefore used ca. 3.5 million windows in training. Figure 7 shows the architecture of the PCN with 7 layers and approximately 250,000 weights to be learned.

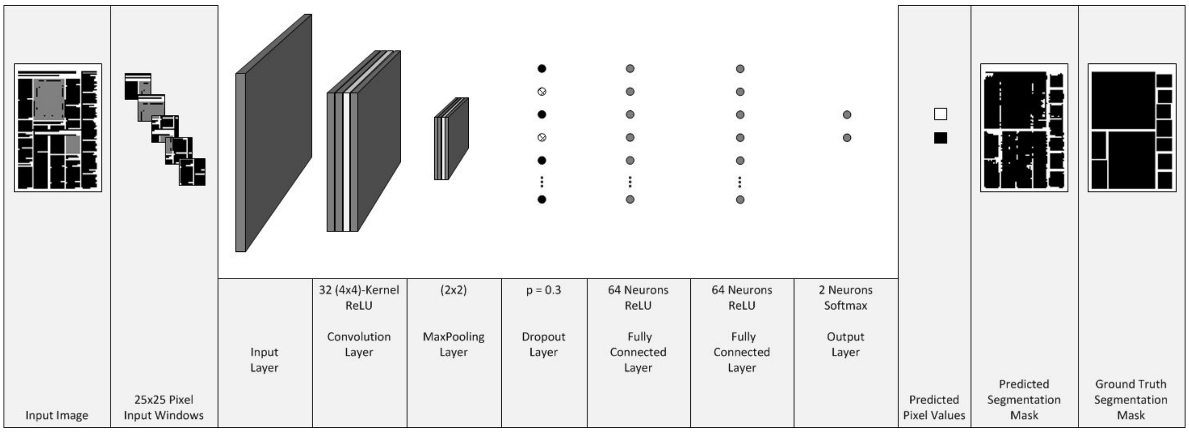
Input Image
25x25 Pixel Input Windows
Input Layer
32 (4x4)-Kernel ReLU Convolution Layer
(2x2) MaxPooling Layer
p = 0.3 Dropout Layer
64 Neurons ReLU Fully Connected Layer
64 Neurons ReLU Fully Connected Layer
2 Neurons Softmax Output Layer
Predicted Pixel Values
Predicted Segmentation Mask
Ground Truth Segmentation Mask

Figure 7. Architecture of the PCN that segments newspaper pages by classifying each pixel of an input image (article or border) using 25x25 pixel windows centered on each pixel to be classified.

The fully convolutional neural network used in our second approach is built with three logical parts (cp. Meier et al. (2017) and Figure 8). Initially, feature extraction is done the same way as with a standard CNN. This is followed by a network performing an upscaling, resulting in a segmentation mask as output. Finally, a very small refinement network adjusts the edges of the article regions (black) to be rectangular, since this is one of the typical characteristics of newspaper pages. We train this network architecture in two steps: first, we run a pre-training with the bigger partially labelled dataset. For this case, the unlabelled parts are replaced by white areas. Second, we use the fully labelled data to finalize the model. For both training steps we insert the original scans together with the transformations as separate channels. Both channels are scaled down to a resolution of 256x256 pixels (keeping the aspect ratio by adding white background where necessary).

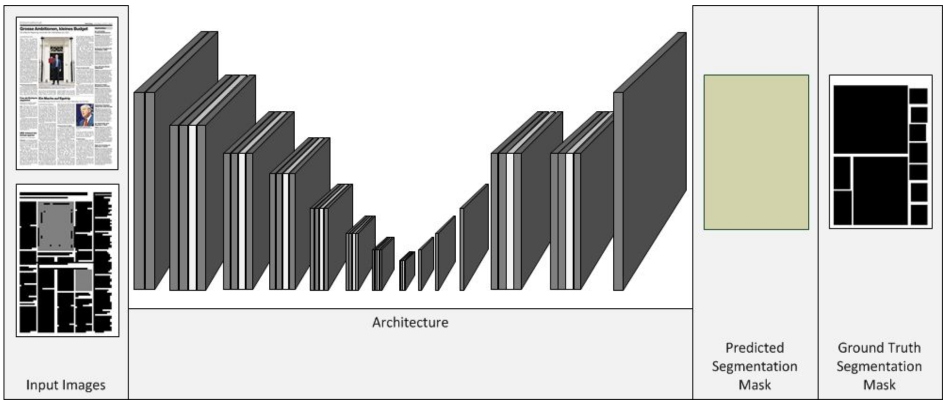

Figure 8. Architecture of the FCN, consisting of three logical parts. First the feature extraction with a standard CNN (up to the center of the figure), second the segmentation (done by upscaling convolutions), and third a refinement network to ensure typical properties of newspaper articles (last block in "Architecture").

4.2 Results

For the evaluation of the PCN and FCN we chose the diarization error rate (DER) known from speaker diarization (Kotti et al., 2008). The DER is a combination of the three error types possible when grouping elements into an unknown number of clusters: confusion error (CE) measures parts of predicted articles that are wrongly assigned; miss error (ME) measures parts of articles that are not included in the predicted segmentations; false alarm error (FE) counts parts of predicted articles that do not overlap with any labelled article from the ground truth.

The FCN has a DER score of 0.1378, thereby outperforming the still respectable PCN (0.2976 DER) by more than 50%. This result shows the impact of a suitable network architecture for the task at hand. While both architectures have comparable runtimes during prediction (roughly 3.7 seconds per page, largely impacted by similar post-processing), the FCN can process images that are approximately 18 times bigger considering that two images are inserted at the same time. On the other hand, while we used around 6,000 pages to train the FCN, we trained the PCN with only 507 pages. We conclude that both approaches can be useful depending on the amount of labelled data that is available. For the given use case, the industrial partner provided the additional 5k partially labeled training images in order to use the FCN approach in practice.

5. Learning to detect outliers: Predictive maintenance with unsupervised deep learning

The condition of critical and costly mechanical equipment is increasingly monitored by observing the vibrations of the machinery under surveillance. In order to detect faults before they damage the whole machinery, traditional methods such as envelope analysis have been used for decades (Randall & Antoni, 2011). However, these methods require knowledge of the machinery's exact geometry and a skilled human operator. An alternative data-driven approach is to automatically detect changes in the signal. This is known as novelty detection and there are plenty of classical methods. For a review, see (Pimentel et al., 2014).

While almost every possible combination of features and classifiers has been tried previously for condition monitoring, the respective literature lacks comparability in terms of data and metrics used as well as given details for reproducibility (Stadelmann et al., 2016). In this case study, we compare several classical novelty detection methods against DL based approaches on a standard bearing data set (Lee et al., 2007), which consist of $n_{train} + n_{test} = 984$ measurements of vibration signals in run-to-failure tests.

As a first step, we use a Fourier transformation to extract $p = 100$ features[13] per measurement to obtain a data matrix $X \in R^{(n_{train}+n_{test}) \times p}$. The details of the feature extraction and the data set can be found in (Fernandez et al., 2013). In the following discussion, we assume that the fault starts to be detectable at frame number 532. This is in line with findings from other researchers (Fernandez et al., 2013) and is also observable from Figure 10, which shows the data matrix (spectrogram).

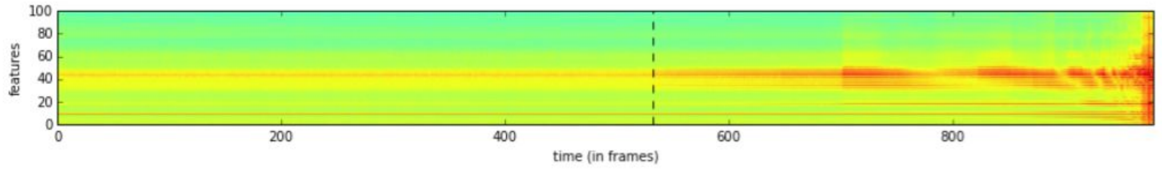

Figure 10. The 100 extracted features (columns) for the 984 time points (rows) for the example data set. The vertical dashed line indicates the first change of the data (frame number 532) as visible by eye.

The output of all methods is a real valued vector of size n_{test} , reflecting the deviation from the normal state learned during the training: the so-called *novelty signal*. All methods are

[13] FFT features: energies of 100 equally spaced frequency subbands, computed over the whole length of the signal (10 seconds).

trained on the first $n_{train} = 200$ rows, where we assume that no fault has occurred. Before training and testing, we apply a robust z-transformation for each feature using the median and the median absolute deviation (MAD) calculated on the training data.

5.1 Classical approaches

We use the following classical methods as baseline:

- One-class SVM (Schölkopf & Smola, 2002) with the variable parameter η, which can be understood as an upper bound of the fraction of outliers.
- Gaussian mixture model (GMM) with a number of $n_{components}$ mixtures (Reynolds & Rose, 1995).
- A simple but robust baseline approach is done in the spirit of the Mahalanobis distance. To reduce the noise, we first transform our data into a n_{comp}-dimensional subspace using PCA with whitening. In that subspace, we calculate the squared Mahalanobis-distance (Bersimis, 2007) to determine the outliers.
- The last classical method in our evaluation uses a PCA learned on the training data, to transform the test data X_{test} into a n_{comp}-dimensional subspace. After that, the data is transformed back into the original space yielding $\hat{X}_{test}$. We use the L^2-based reconstruction error as the novelty signal. This corresponds to an autoencoder without nonlinearities and with tied weights.

Figure 11 shows the results of the classical methods described above.

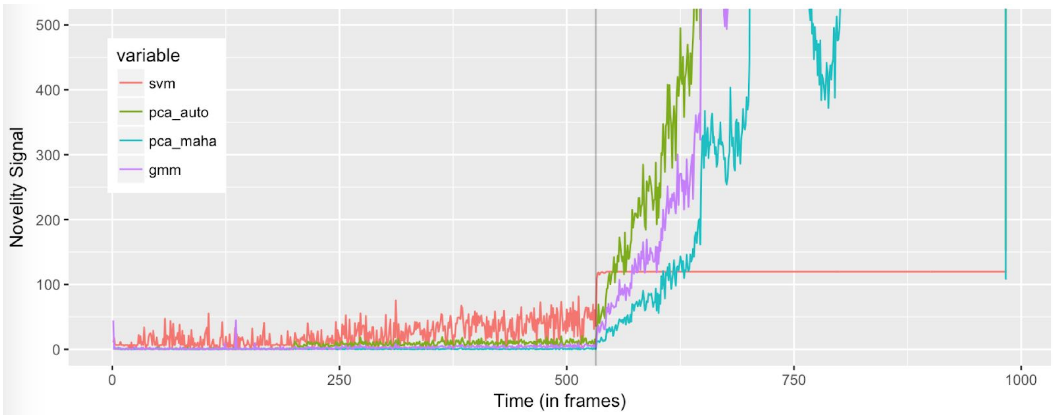

Figure 11. Novelty signal for the classical methods. The used parameters are $\eta = 0.1$ for the SVM, $n_{components}$=16 for GMM and n_{comp}=50 for the PCA-based methods. To focus on the sudden change of the novelty signal at 532, the maximum of the y-axis has been limited to 500 (maximum signal is in the order of 1E10).

All methods show an amplitude increase in the novelty signal at frame number 532, where we assume that the fault is detectable. The `pca_auto` method shows the strongest increase in the signal and is used later for the comparison with the deep learning based methods.

5.2 Deep learning based methods

After using the classical methods to establish a baseline, we now consider deep autoencoders. All methods are trained for 50 epochs with a batch size of 20. We start with a simple fully connected autoencoder using sigmoids as activations. A detailed description of a fully connected autoencoder can be found e.g. in (Goodfellow et al., 2016). We investigated different numbers of hidden layers and determined that for 5 hidden layers there is the best compromise between steepness of the novelty signal after the fault and the noise before it (see `fc_auto` in Figure 12).

In addition to the `fc_autoencoder`, we also include a recurrent version, in which the neurons are replaced by Long Short-Term Memory cells (LSTMs) (Hochreiter and Schmidhuber, 1997). We found that an architecture with 3 hidden layers consisting of 10, 4, and 10 nodes, respectively, performed best. These results are displayed in Figure 12. The behavior is similar to the standard autoencoder and suggests that the temporal ordering of frames is unimportant here.

The final two autoencoders introduce means for additional regularization. The first one, the denoising autoencoder, does this by injecting additional noise, see (Vincent et al., 2010) for details. The best performance was observed with 10 nodes in the hidden layer. The second one is the variational autoencoder (VAE) (Kingma and Welling, 2013). Its optimal architecture turned out empirically to have 32 nodes in the hidden layer and a 4-dimensional latent space, which is shown in Figure 12, labeled as `vae_32_4`. Note that, in principle, the VAE can also be extended to generate novel data.

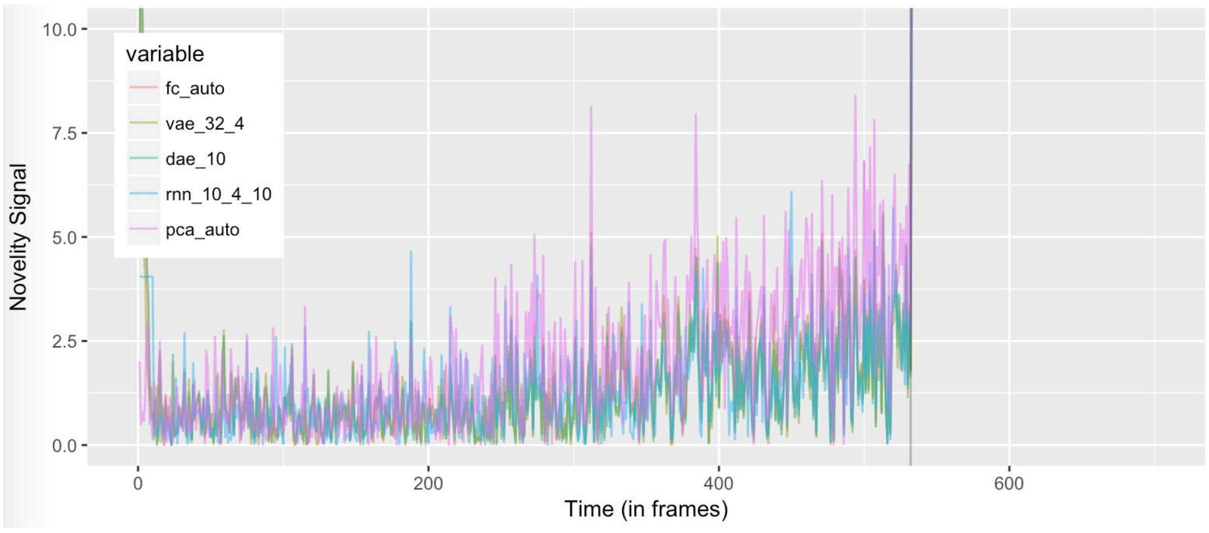

Figure 12: Novelty signal for the deep learning based methods and the best classical approach (`pca_auto`). All methods show a steep ascent after the fault is detectable at frame number 532. To better illustrate the signal before the fault, we limited the range of the normalized novelty signal to [0, 10].

In conclusion, all methods (classical and DL) show a similar novelty signal and detect the fault at time frame 532. However, the DL based methods give a weaker novelty signal in a region where there is no fault. Here, the best classical method (`pca_auto`) shows a stronger signal at times before the fault occurred. We conclude that the given task is too simple to profit from the more capable models - DL is not needed on this specific data set.

6. Lessons Learned

Deep learning approaches have proven useful not only in academic computer vision settings, but also in various scenarios inspired by real business use cases: We have improved the state of the art in high content screening, speaker clustering, and automatic article segmentation, while showing at least comparable results for condition monitoring. Overall, the authors have verified the practicability of DL applications on at least 10 substantial research projects in collaboration with industry during the last four years. Contrary to public opinion, Szegedy et al. (2014) note that *“most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a* *consequence of new ideas, algorithms and improved network architectures"*. This is according to our experience worth considering.

6.1 Working with limited resources

Our biggest take-home message is the importance of working well with limited resources. Having a good set of data for training and evaluation (i.e., available at the start of the project, ideally large[14], in a good shape for further processing, resembling the true distribution of the problem to be solved) is the starting point: it doesn't pay off to "negotiate" minimum numbers of needed data with business owners. Rather, "the more the better" is key. If the most one can get is still little, the following tricks may apply:

- Using available pre-trained networks that have been trained for a "close enough" task (like e.g. the VGG-16 network[15] for any image classification task) to do transfer learning.
- Use trainable architectures like Inception (Szegedy et al., 2014) or Resnet (He at al., 2015b) that adapt their complexity to the available data and may even be compressible (Han et al., 2015).
- Do sensible data augmentation (see Section 2): provide the training procedure with variants of your original data that (a) you can create randomly on the fly and that (b) resemble distortions / alterations relevant and realistic in practice.
- Often there is enough unlabeled data, but labeling is costly. In that case one can try to employ semi-supervised learning methods, which are currently being actively developed (Kingma et al., 2014). Another possibility is to use high-level features created by a first network to do a clustering or t-SNE embedding similar to Figure 5 (see Section 3). This allows to label lots of data after a short inspection.

Sometimes, data is not the limiting factor, but hardware is (at least for applying the trained model later). While compressed networks help to speed up network application considerably, it should be noted that while neural network training of practically relevant size may take weeks on dedicated hardware (i.e., latest generation of GPU workstations), the application might be doable in real time even on embedded devices like a raspberry pi[16] (see also Section 4). And as Section 5 has shown, DL approaches might not always outperform simple baseline approaches; so it always pays off to compare against classical methods (at least to establish a benchmark, see Section 2).

6.2 Other advice

Additional advice can be summarized as follows:

- Having a good start on a new use case often depends on (a) starting from an easy, well understood baseline model closely resembling a published architecture and task[17], and (b) to slowly increase the complexity of the architecture. As a rule of thumb, if a human can see/hear/... the solution to a pattern recognition problem in the training data, it can be extracted using machine learning algorithms (Domingos, 2012).
- If it is not a standard problem, ensure to provide a loss function which really describes the problem that is going to be solved (see section 3).

[14] Personal rule of thumb of one of the authors (T.S.): I feel comfortable with a small to medium four-digit number of instances per class in a classification setting.

[15] http://www.robots.ox.ac.uk/~vgg/research/very_deep/

[16] https://www.martinloeser.eu/deutsch/forschung/pivision/

[17] Find a collection of models per task e.g. here: https://github.com/sbrugman/deep-learning-papers

- Latest algorithmic developments in neural nets like dropout or batchnorm, ADAM / ADADELTA and ReLU are “always on” in our projects if applicable[18] as they considerably ease training to the point that makes applications possible that just do not work without them (see Section 3).
- It is common that a first instance of a DL model does not work on a completely new task and data set. Then, debugging is key, ranging in methodology from checking for the application of best practices[19], hand-calculating the training equations for toy examples (to find implementations problems e.g. in the loss function[20]), visualizing the pre-processed data (to see if data loading might be buggy) or learned weights[21] and inspecting loss values (does it learn at all[22]?) as well as misclassified training examples (to get intuition into what goes wrong (Ng, 2018)).
- The speed of new advances in DL is breathtaking at the moment. While new developments are published daily on arXiv[23], news aggregators like reddit[24] or Data Machina[25] and explanation-focused journals like Distill[26] help to stay up-to-date. For a real project, it is important to check the current state of the art at least back to the latest major DL conferences NIPS[27], ICML[28] and ICLR[29] and the trends discussed there in tutorials and keynotes: paradigms are still evolving, and new applications are shown daily.

General best practices for DL applications are also summarized by Ng (2016), Hinton et al. (2012) and LeCun et al. (1998b).

Acknowledgements

The authors are grateful for the support by CTI grants 17719.1 PFES-ES, 17729.1 PFES-ES and 19139.1 PFES-ES.

7. References

Aucouturier, J.-J., Defreville, B., & Pachet F. (2007). The bag-of-frames approach to audio pattern recognition: a sufficient model for urban soundscapes but not for polyphonic music. *J Acoust Soc Am.* 2007 Aug, 122(2), pp. 881-91.

Beigi, H. (2011). Fundamentals of speaker recognition. *Springer Science & Business Media*.

Bersimis, S., Psarakis, S., & Panaretos, J. (2007). Multivariate statistical process control charts: an overview. *Quality and Reliability Engineering International*, vol. 23, pp.517–543.

Bishop, C. M. (2006). Pattern recognition and machine learning. *Springer*.

[18] For example, dropout does not work with ResNets and has largely been replaced by batchnorm in these architectures (Zagoruyko & Komodakis, 2016).
[19] See also https://engineering.semantics3.com/2016/10/09/debugging-neural-networks-a-checklist/
[20] See also https://gab41.lab41.org/some-tips-for-debugging-deep-learning-3f69e56ea134
[21] See e.g. https://distill.pub/2017/feature-visualization/ and https://github.com/bruckner/deepViz
[22] See also http://russellsstewart.com/notes/0.html
[23] https://arxiv.org/
[24] https://www.reddit.com/r/MachineLearning/
[25] https://www.getrevue.co/profile/datamachina
[26] http://distill.pub/
[27] https://nips.cc/
[28] http://icml.cc
[29] http://iclr.cc

Bouma, G. (2009). Normalized (pointwise) mutual information in collocation extraction. In: From form to meaning: processing texts automatically, *Proceedings of the Biennial GSCL Conference* 2009, pp. 31--40,
https://svn.spraakdata.gu.se/repos/gerlof/pub/www/Docs/npmi-pfd.pdf

Chung, J. S., Senior, A. W., Vinyals, O., & Zisserman, A. (2016). Lip reading sentences in the wild. *CoRR*, vol.1611.05358.
http://arxiv.org/abs/1611.05358

Ciresan, D., Meier, U., Masci, J., & Schmidhuber, J. (2012a). Multi-column deep neural network for traffic sign classification. 2012.
http://people.idsia.ch/~juergen/nn2012traffic.pdf

Ciresan, D., Giusti, A., Gambardella, L. M. & Schmidhuber, J. (2012b). Deep neural networks segment neuronal membranes in electron microscopy images. *Advances in Neural Information Processing Systems 25*, pp. 2843-2851.
http://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-images.pdf

Dehak, N., Kenny, P., Dehak, R., Dumouchel, P., & Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798.
http://habla.dc.uba.ar/gravano/ith-2014/presentaciones/Dehak_et_al_2010.pdf

Dieleman, S., & Schrauwen, B. (2014). End-to-end learning for music audio. In *Proceedings of ICASSP*, pp. 6964–6968.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, vol. 55, no. 10, pp. 78-87.

Dürr, O., Duval, F., Nichols, A., Lang, P., Brodte, A., Heyse, S., Besson, D. (2007). Robust hit identification by quality assurance and multivariate data analysis of a high-content, cell-based assay. *Journal of biomolecular screening* vol. 12, np. 8, pp. 1042-1049

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (KDD-96). AAAI Press. pp. 226–231.

Fukushima, K. (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. 36 (4): 193–202. doi:10.1007/BF00344251. PMID 7370364.

Ganchev, T., Fakotakis, N., & Kokkinakis, G. (2005). Comparative evaluation of various MFCC implementations on the speaker verification task. In *Proceedings of SPECOM 2005*, vol. 1, pp. 191–194.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT Press*.
http://www.deeplearningbook.org

Gustafsdottir, S. M. , Ljosa, V., Sokolnicki K. L., Wilson, J. A., Walpita, D., Kemp, M. M., Petri Seiler, K., Carrel, H. A., Golub, T. R., Schreiber, S. L., Clemons, P. A., Carpenter, A. E., & Shamji, A. F. (2013). Multiplex cytological profiling assay to measure diverse cellular states. *PLoS ONE* 12, e80999.

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. *CoRR*, vol. 1510.00149.
https://arxiv.org/abs/1510.00149

Hinton, G. E., Srivastava, N., & Swersky, K. (2012). Lecture 6a: overview of mini-batch gradient descent. In *"Neural Networks for Machine Learning"*, University of Toronto.
https://www.coursera.org/learn/neural-networks

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. *CoRR*, vol. 1502.01852.
http://arxiv.org/abs/1502.01852

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Deep residual learning for image recognition. *CoRR*, vol. 1512.03385.
https://arxiv.org/abs/1512.03385

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*. 9 (8) pp. 1735–1780.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, vol. 148, pp. 574--591.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/

Hsu, Y.-C., & Kira, Z. (2015). Neural network-based clustering using pairwise constraints. *CoRR*, vol. 1511.06321.
https://arxiv.org/abs/1501.03084

Ioffe S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, vol. 37, pp. 448--456.
https://arxiv.org/pdf/1502.03167

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. *CoRR*, vol. 1312.6114.
https://arxiv.org/abs/1312.6114

Kingma, D. P., & Ba, J. (2014). Adam: a method for stochastic optimization. *CoRR*, vol. 1412.6980, 2014.
http://arxiv.org/abs/1412.6980

Kingma, D. P., Mohamed, S., Rezende, D. J., & Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pp. 3581-3589.
https://papers.nips.cc/paper/5352-semi-supervised-learning-with-deep-generative-models

Kotti, M., Moschou, V., & Kotropoulos, C. (2008). Speaker segmentation and clustering. *Signal Processing*, vol. 88, issue 5, pp. 1091–1124.
http://dx.doi.org/10.1016/j.sigpro.2007.11.017

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25, pp. 1097-1105.
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998a).Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86 (11), pp. 2278–2324. http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

LeCun, Y., Bottou, L., Orr, G. B., & Mueller, K.-R. (1998b). Efficient BackProp. In G. B. Orr and K.-R. Mueller, Neural Networks: Tricks of the Trade, *Lecture Notes in Computer Science* vol. 1524, pp. 9-50.

LeCun, Y., Bengio Y., & Hinton, G. E. (2015). Deep learning. *Nature*, 521, pp. 436–444.
http://www.nature.com/nature/journal/v521/n7553/full/nature14539.html

Lee, C.-Y., & Osindero, S. (2016). Recursive recurrent nets with attention modeling for OCR in the wild. *CoRR*, vol. 1603.03101.
https://arxiv.org/abs/1603.03101

Lee, J., Qiu, H., Yu, G., & Lin, J. (2007). Bearing data set. IMS, *University of Cincinnati*, NASA Ames Prognostics Data Repository, Rexnord Technical Services.
https://ti.arc.nasa.gov/tech/dash/pcoe/prognostic-data-repository/

Ljosa, V., Sokolnicki, K. L., & Carpenter, A. E. (2009). Annotated high-throughput microscopy image sets for validation. *Nat. Methods* 9, 637.

Long, J., Shelhamer, E., & Darrell, T. (2014). Fully convolutional networks for semantic segmentation.
https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Lukic, Y. X., Vogt, C., Dürr, O., & Stadelmann, T. (2016). Speaker identification and clustering using convolutional neural networks. In *Proceedings of IEEE MLSP* 2016.

Lukic, Y. X., Vogt, C., Dürr, O., & Stadelmann, T. (2017). Learning embeddings for speaker clustering based on voice quality. In *Proceedings of IEEE MLSP* 2017.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*. 1. University of California Press. pp. 281–297.

Meier, B., Stadelmann, T., Stampfli, J., Arnold, M., & Cieliebak, M. (2017). Fully convolutional neural networks for newspaper article segmentation. In *Proceedings of ICDAR 2017*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pp. 3111–3119.
https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical Report, *Rutgers University*, New Brunswick, New Jersey, USA.
http://www.cs.nott.ac.uk/~pszbsl/G52HPA/articles/Mitchell:80a.pdf

Moravcik, M., Schmid, M., Burch, N., Lisy, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., & Bowling, M. H, (2017). DeepStack: expert-level artificial intelligence in no-limit poker”, *CoRR*, vol.1701.01724.
http://arxiv.org/abs/1701.01724

Mori, S., Nishida, H., & Yamada, H. (1999). Optical character recognition. *John Wiley & Sons*, New York, NY, USA, ISBN 0471308196.

Nielsen, F. A. (2017). Status on human vs. machines, post on *“Finn Årup Nielsen's blog”*.
https://finnaarupnielsen.wordpress.com/2015/03/15/status-on-human-vs-machines/

Nielsen, M. A. (2015). Neural Networks and Deep Learning. Determination Press.
http://neuralnetworksanddeeplearning.com

Ng, A. (2016). Nuts and bolts of building AI applications using deep learning. *NIPS* Tutorial.

Ng, A. (2018). Machine learning yearning. (to appear).
http://www.mlyearning.org/

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp.1345-1359.
http://ieeexplore.ieee.org/abstract/document/5288526/

Pimentel, M. A. F., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014). A review of novelty detection, *Signal Processing*, pp.215–249.

Randall, R. B., & Antoni, J. (2011). Rolling element bearing diagnostics—a tutorial. *Mechanical systems and signal processing*, vol. 25, no. 2, pp. 485-520, Elsevier.

Razavian, A. S., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. *CVPR 2014*, pp. 806-813.
https://arxiv.org/abs/1403.6382

Reynolds, D. A., & Rose, R. C. (1995). Robust text-independent speaker identification using Gaussian mixture speaker models. *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72-83.

Reynolds, D. A., Quatieri, T. F., & Dunn, R. B. (2000). Speaker verification using adapted gaussian mixture models. *Digital signal processing*, vol. 10, no. 1, pp. 19–41.

Romanov, A., & Rumshisky, A. (2017). Forced to learn: discovering disentangled representations without exhaustive labels”. *ICRL 2017*.
https://openreview.net/pdf?id=SkCmfeSFg

Rosenblatt, F. (1957). The Perceptron - a perceiving and recognizing automaton. Technical report 85-460-1, *Cornell Aeronautical Laboratory.*

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Neurocomputing: Foundations of Research*, pp. 696--699, MIT Press.
http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf

Schmidhuber, J. (2014). Deep learning in neural networks: an overview.
https://arxiv.org/abs/1404.7828

Schölkopf, B., & Smola, A. J. (2002). Learning with kernels: support vector machines, regularization, optimization, and beyond. *MIT press*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, vol. 529, pp 484--503.
http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, vol. 1409.1556.
https://arxiv.org/abs/1409.1556

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol.15, no.1, pp. 1929-1958.
http://dl.acm.org/citation.cfm?id=2627435.2670313

Stadelmann T., & Freisleben, B. (2009). Unfolding speaker clustering potential: a biomimetic approach. In *Proceedings of the 17th ACM international conference on Multimedia*. ACM, pp. 185–194.

Stadelmann, T., Musy, T., Duerr, O., & Eyyi, G. (2016). Machine learning-style experimental evaluation of classic condition monitoring approaches on CWRU data. Technical report, *ZHAW Datalab*, (unpublished).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, vol. 1409.4842.
https://arxiv.org/abs/1409.4842

Szeliski, R. (2010). Computer vision: algorithms and applications. *Texts in Computer Science*, Springer-Verlag New York.
http://szeliski.org/Book/

van der Maaten, L., & Hinton, G. E. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579– 2605.
http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11., pp. 3371-3408.
http://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf

Weyand, T., Kostrikov I., & Philbin, J. (2016). PlaNet - photo geolocation with convolutional neural networks. *CoRR*, vol. 1602.05314.
http://arxiv.org/abs/1602.05314

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., & Zweig, G. (2016). Achieving human parity in conversational speech recognition". *CoRR*, vol. 1610.05256.
http://arxiv.org/abs/1610.05256

Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks. In Wilson, R.C., Hancock, E.R., & Smith, W.A.P. (editors), Proceedings of the British Machine Vision Conference (BMVC), pp.87.1-87.12. BMVA Press.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, vol. 1212.5701.
http://arxiv.org/abs/1212.5701

Zheng, F., Zhang, G., & Song, Z. (2001). Comparison of different implementations of MFCC. *Journal of Computer Science and Technology*, vol. 16, no. 6, pp. 582--589.
http://dx.doi.org/10.1007/BF02943243