Proceedings of Machine Learning Research 74:92–102, 2017 Full Paper - LIDTA 2017

Predicting Defective Engines using Convolutional Neural Networks on Temporal Vibration Signals

Nikou Günnemann NIKOU.GUENNEMANN@TUM.DE
Jürgen Pfeffer JUERGEN.PFEFFER@TUM.DE
Technical University of Munich, Germany

Editors: Luís Torgo, Bartosz Krawczyk, Paula Branco and Nuno Moniz.

Abstract

This paper addresses for the first time the problem of engines' damage prediction using huge amounts of imbalanced data from "structure borne noise" signals related to the internal engine excitation. We propose the usage of a convolutional neural network on our temporal input signals, subsequently combined with additional static features. Using informative mini batches during training we take the imbalance of the data into account. The experimental results indicate good performance in detecting the minority class on our large real-world use case.

Keywords: Prediction, Predictive Maintenance, Imbalanced Data, Deep Learning.

1. Introduction

Combustion engines are becoming increasingly complex to meet a variety of standards. To guaranty a smooth assembly process and functional engines, an efficient cost effective cold test is performed at the end of each production line. During the cold test, the engine is passively driven by an electric motor. In doing so, several measurements including structure bornes – mechanical vibrations at lower frequencies as 50 **Hz** or 100 **Hz** – are measured for determining the properties of the materials of the engines (Cremer and Heckl, 2013). Based on these structure-borne measurements, the goal of our work is to predict the result of the endurance run for the tested engine: is the engine fully functional or defective?

The collected measurements, however, implicate various challenges, making the prediction task very hard: (1) The available signals collected from different process steps refer to different views. While the structure-borne measurements are collected during the cold test, other views such as the engine information in general or environment features are additionally considered. Such 'multi-view' data are of different types followed by different statistical distributions and different kinds of uncertainties (Sun, 2013). (2) Furthermore, the high dimensionality of the collected signals pose a huge challenge: each signal itself represents a time series. It is worth point out that the lengths of these time series differ between signals. Thus, it can not simply be treated as a multivariate time series. (3) Most importantly, the data suffers from a highly imbalanced distribution of class labels. That is, the class of defective engines is highly under-represented compared to the class of non-defect ones which makes learning quite a difficult task.

Motivated by the effectiveness of deep learning techniques in recent years, we study in this work – using data from a real world use case – whether such architectures also enable to predict defective engines from the collected imbalanced data.

2. Related Work

Deep learning has achieved extremely high performance in many tasks in the field of image or video recognition (Schmidhuber, 2015; Karpathy et al., 2014) as well as natural language processing (Mikolov et al., 2013). Likewise, researchers have investigated deep neural network architectures and algorithms to evaluate their performance in time series tasks – particularly, real world applications with focus on multivariate time series classification have become increasingly important, such as in health care, human activity recognition, or renewable energies (Doucoure et al., 2016; Yang et al., 2015; Zheng et al., 2014; Borovykh et al., 2017).

Since neural networks are capable of extracting non-linear information from data they are a natural fit for time series which rarely represent only linear effects. While on a first sight, temporal/sequence neural network models such as RNNs/LSTMs seem to be a natural choice for temporal data classification, the early work of Kim (2014) has shown that simple Convolutional Neural Networks (CNNs) are equally or even better suited for such tasks. With this idea in mind, (Cui et al., 2016; Borovykh et al., 2017) have shown that CNNs perform impressively not only in classical use cases like face verification and audio classification but also in tasks like time series classification.

The main reason for the success of CNN in this context is explained by its ability to learn from local information, being correlated to further features by sliding the convolutional filter along time axis. Denoting the convolution as the main advantage of this model, in this paper we focus on the questions: (a) Having a real-world multi-view dataset with high dimensionality, is a CNN architecture capable of predicting engines affected by malfunctions with a high Area Under the Curve (AUC)? (b) Can CNNs perform well even when the data is massively imbalanced?

To answer these questions, we investigate a CNN deep learning architecture (Section 4) to predict defective engines using data from a real-world use case (Section 3). We investigate different architectures of the CNN, including different number of layers, using different pooling functions and activation functions in our experimental study – as well as different properties of the data and its effect on the result (Section 5).

3. Data Description

To gather the structure borne noise of each engine, a vibrometer is used to measure different mechanical surface vibrations in time domain. Besides, the influence of the engine vibration on the air pressure is measured. Using fourier transformation, the measured signals are converted in sequential frequency domain. Figure 1 shows a snippet of 3 measured signals for *one* engine. While the x-axis shows different sequence of frequency spectra, the y-axis presents the measured values pointing to e.g. a vibration signal or crankshaft motions etc.

Formally, the resulting data is a set of matrices $S_i \in \mathbb{R}^{N \times L_i}$, where each $i \in M$ represents *one* signal measured for all instances. Here, N denotes the number of instances/engines and

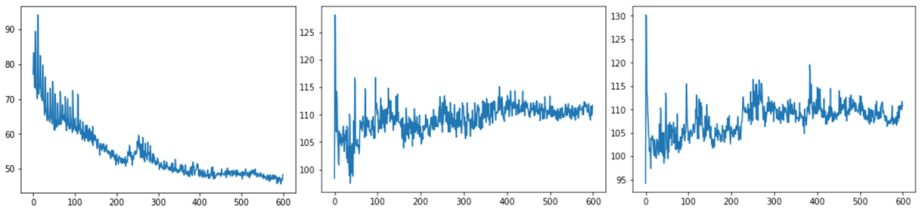

Figure 1: Visualization of three different signals for one engine.

L_i the length of frequency spectra regarding signal i. Accordingly, the jth row of S_i refers to the frequency spectrum collected for engine $j \in N$ regarding signal i. For simplicity we denote these row vectors with x_j^i. Note again that for each signal i the length of these vectors L_i might differ.

Furthermore, for each engine we have a set of non-sequential (environmental) variables, which are indicated by the matrix $E \in \mathbb{R}^{N \times L_E}$ – with e_j denoting the jth row vector. And each instance $j \in N$ is assigned to a label y_j with $y_j \in \{0, 1\}$. Here, $y_j = 1$ represents class of instances with a certain issue and $y_j = 0$ the opposite. We denote with N_1 all instances having label 1, and N_0 all instances with label 0.

Overall, for our use case we have $\approx 93,000$ engines. For each engine, 6 different sequential based signals complemented with 7 environment features are considered. The ratio of defected instances compared to non-defective ones is around 1:32.

It is worth noting that the signals of functional and defect instances cannot be distinguished easily, making the prediction very difficult and not possible by humans. It is also noticeable that the illustrated signals include strong fluctuations with several local minima and maxima. By using CNNs on S_i, our goal is to exploit this information to perform accurate predictions.

4. Convolutional Neural Network on Engine Vibration Signals

In the following we will describe the structure of our deep CNN used to classify an engine as a malfunction one or not.

4.1. Deep CNN Architecture

Figure 2 shows the general structure of the deep feedforward neural network used in our proposed model.

(A) First, to facilitate the training of CNN layers and simultaneously to improve the efficiency, straight after signal processing, we first down-sample our signals by applying a (1x5) Max-pooling. In doing so, we progressively reduce the spacial size of the amount of features and computation complexity while keeping the location sensitivity in the model since the size of pooling is small.

(B) Second, we use multiple CNN/pooling layers as non-linear learnable feature extractors on the temporal signals. More precise, we combine the following three functions whose variations we investigate in our experimental analysis:

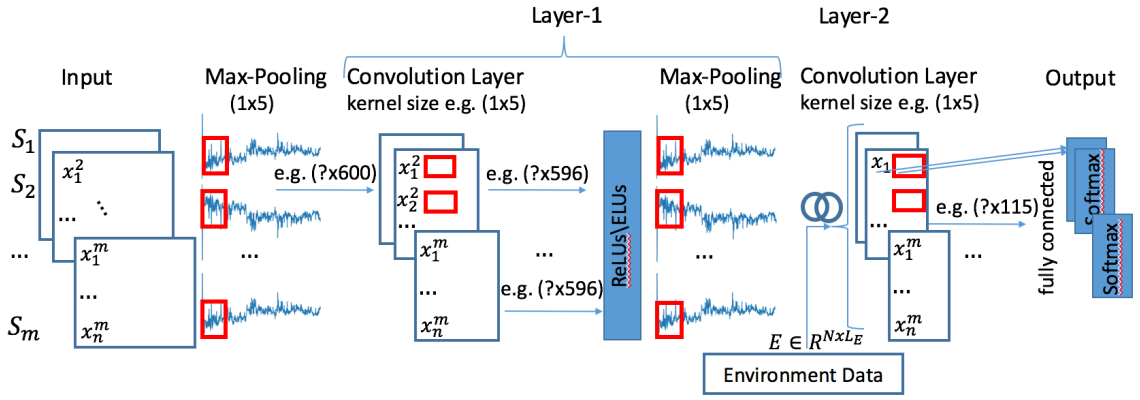

Figure 2: An overview of the two layered CNN-architecture: as input it obtains the engine signals which are concatenated with environment data on a later layer

(i) Convolutional layer: It convolves learnable kernel filters across the input data S_i to compute a dot product between the entries of the filter and the input. For each signal (and layer) we learn different kernel weights, thus, accounting for the multi-view nature of the data. As a result, the network learns different filters which aim to detect specific type of features at certain positions. To extract maximum local temporal information from the whole data, it is beneficial to restrict the size m_l of the filters f_i^l, where l is the corresponding layer and i the ith signal. Note that in contrast to, e.g., images our filters are one-dimensional (i.e. vectors).

(ii) Activation layer: The results generated by the convolutional layer is given as an input to the *activation function* to capture some of the non-linearities from the input data. In our model we use the exponential linear unit *(ELU)* activation function (Clevert et al., 2015). While generally the most used activation function is the rectified linear units (ReLUs) Nair and Hinton (2010), it turned out that ELUs are more powerful in our scenario (see experimental study). By mapping negative values to zero, ReLU are helpful in leading to less dense solutions. However, as a disadvantage the phenomena of *dead* ReLU can be observed – e.g. in the case when having a large negative bias term from the previous layer.

One attempt to address this issue is the Exponential Linear Units (ELUs), defined as:

$$f(x) = \begin{cases} x_i^{l-1}, & \text{if } x_i^{l-1} > 0 \\ \alpha(\exp(x_i) - 1), & \text{if } x_i^{l-1} \leq 0 \end{cases} \tag{1}$$

In contrast to ReLUs, ELU can deal with negative values of feature maps from convolutional layers (Clevert et al., 2015). This is realized by α with $\alpha > 0$. The negative values of feature maps push the mean activations closer to zero which enable faster learning for bringing the gradient closer to the natural gradient. When the input gets smaller, ELUs saturate to a negative value. This decreases the variation and the information propagate to the next layer. The performance of both functions if analyzed in Section 5.

(iii) Pooling Layer: Finally, we down-sample the resolution of the input from the previous layer using either a) the average value in each neighborhood within the selected

pooling window or b) the maximum value. The intuition is, that the exact location of features is not important as its rough location relative to other features Zheng et al. (2014). Note that the last convolutional layer is not followed by a pooling. We analyzed the effects of different pooling window and convolution sizes in our experimental study.

(C) Last, after performing potentially multiple CNN/pooling layers (operating on each temporal signals) we combine the output with the static environmental data by feeding them jointly into a fully connected neuron – followed by a softmax activation function to generate the final classification decision.

More precise, let $\hat{x}_j^i \in \mathbb{R}^{k_i}$ be the output of the last CNN layer (for engine j and signal i) and e_j the environmental features of engine j, the predicted probability of being a malfunction engine is given by

$$\tilde{y}_j = \frac{1}{1 + exp(-w^T \cdot \delta_j + b)} \quad \text{where} \quad \delta_j = [\hat{x}_j^1, \hat{x}_j^2, \ldots, \hat{x}_j^M, e_j]$$

Here $w \in \mathbb{R}^{L_E + \sum_{i=1}^{M} k_i}$ and $b \in \mathbb{R}$ are learnable weights. Since we consider here a two-class problem, the probability of being a correctly functional engine is accordingly $1 - \tilde{y}_j$.

4.2. Loss function, class imbalance, and batch learning

To learn the parameters of the neural network, we have to define a corresponding loss function – operating on the predicted class probabilities. Naively, in a (binary) classification task one could employ the cross entropy loss:

$$L_1(\{\tilde{y}_i\}_{i \in N}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \cdot log(\tilde{y}_i) + (1 - y_i) log(1 - \tilde{y}_i)] \tag{2}$$

where y_i denotes the true label and $\tilde{y}_i$ the predicted probability of the instance i, respectively. By minimizing the loss – e.g. using gradient descent –, our goal is to lower the prediction error with the hope to keep the AUC on the test set as high as possible.

Class Imbalanced Problem However, the engine dataset analyzed in this paper is hit by the class imbalance problem (Japkowicz and Stephen, 2002), where number of positive instances (i.e. the number of malfunctioned engines in our case), is clearly less than the total number of non-damage engines. When the number of instances from one class arises the other one, the learning process from the training data is adversely effected. Due to the importance of this issue, many existing research works have addressed this problem (Bermejo et al., 2011; Mazurowski et al., 2008). In general, these approaches can be divided into three main classes (Galar et al., 2012): ***(a) Algorithm level approaches*** try to modify the learning algorithm to bias the learning with focus on the even class as the minority one. Required is here a prior know how why the learning behaviour of the algorithms fails when e.g. the class of damaged engines is in minority. ***(b) Data level approaches*** rebalance the dataset by resampling the even class and the non even class in e.g. an equal distribution (Stefanowski and Wilk, 2008). ***(c) Cost sensitive learning*** is a combination of the first two versionsa) and b). While the focus is to assign costs to different instances to e.g. prioritize them in the minority class, the classifier is biased toward the even class

by minimizing e.g. the result of the loss function used in classification (Maldonado et al., 2014; Günnemann and Pfeffer, 2017).

In this paper, we tackled the class imbalance problem by using a combination of data level and cost sensitive approaches. More precise, we used stochastic learning (mini batches) to not only speed up the learning process but also to handle imbalance data.

First, we aim to minimize the following loss function

$$L_2(\{\tilde{y}_i\}_{i\in N}) = -\frac{1}{N}\sum_{i=1}^{N}[\alpha \cdot y_i \cdot log(\tilde{y}_i) + (1-y_i)log(1-\tilde{y}_i)] \tag{3}$$

where $\alpha = \frac{|N_0|}{|N_1|}$ is the ratio of class 0 to 1 (in our data: $\alpha \approx 32$). That is, class 1 is α times less frequent than class 0. Intuitively, using Eq. 3, the less frequent class gets higher influence.

Second, we minimize the loss function using stochastic gradient descent using *informative* mini batches. Instead of randomly selecting instances for the minibatch, each minibatch B contains *all* instances N_1 and a random subset of instances from N_0 of the same carnality. Thus, the batch B is 'biased' towards the minority class (compared to the original data distribution). This is specifically helpful if the minority class is very small in the overall data since the likelihood to pick these instances in a minibatch would be small.

Using the informative batches B, the stochastic gradient descent step is now computed based on the loss L_1 (not L_2): $L_1(\{\tilde{y}_i\}_{i\in B})$. The class imbalance factor α is already absorbed by the minibatch. Thus, overall we not only deal with class imbalance problem but also speed up the learning process by training our CNN on a stochastic sample.

5. Experimental Results and Discussions

As mentioned in Section 3, the performance of proposed CNN network was tested on a structure born noise dataset consisting of multiple signals, each in turn, representing 92890 frequency spectra vectors. In our case, each vector presents an engine mapped to a binary label. Using the proposed CNN architecture on our dataset, our goal is to classify the two groups of damaged respectively no damaged engines from each other with a high AUC. We will now analyze our model in more details. All experiments were performed using a train-test split of 80:20 and learning was done using the ADAM optimizer until convergence. All numbers indicate the results on the test set. The model was implemented in TensorFlow and executed on a K80 GPU disposing of 6 cores and 56GB Memory.

5.1. Analysis of parameters

We start by assessing the performance of our model when varying different parameters of the architecture and learning method.

Window sizes and number of layers. Table 1 summarizes different AUCs produced by using different sizes of convolutional filters or pooling windows used in our model – as well as the number of layers used in the architecture (see Fig 2). As described, a convolution layer is followed by a pooling layer unless the last layer of the model. For a fair comparison, we used the same activation function (ELUs), pooling function (Max-function), and learning rate 0.0001 for all architectures.

nr.	layer 1		layer 2		layer 3		auc
	conv	pool	conv	pool	conv	pool	
1	(1,3)	(1,3)	(1,3)	-	-	-	0.889
2	(1,5)	(1,5)	(1,5)	-	-	-	0.913
3	(1,3)	(1,5)	(1,3)	-	-	-	0.90
4	(1,3)	(1,10)	(1,3)	-	-	-	0.90
5	(1,5)	(1,10)	(1,5)	-	-	-	0.91
6	(1,3)	(1,3)	(1,3)	(1,3)	(1,3)	-	0.87
7	(1,5)	(1,5)	(1,5)	(1,5)	(1,5)	-	0.89

Table 1: Impact of size of filters and poolings on AUC.

For the convolution layer, we consider the two filters of size (1x3) and (1x5) since any bigger size would cause loss of information in the temporal domain. In addition to this, we analyzed the influence of max-pooling windows of size (1x3), (1x5) and (1x10) on AUC values. Considering the results of the 2 layers architectures, it is obvious that using pooling and filter with size of (1x5) leads to the best result followed by a pooling size of (1x10), where AUC value drops slightly.

Considering the results of the 3 layers' architecture, a better result is not achieved. One reason might be the larger number of parameters, which makes the learning approach more difficult and also might lead to overfitting. Therefore, in the following experiments, we focus on the two layer architecture with a window size of (1x5) for pooling and filtering.

Learning rate and convergence. We also evaluated the AUC of our model in dependency of the learning rate parameter. From the results in Figure 3, we can see that by setting the value of the learning rate denoted by lr equal to 0.01, the loss value fluctuates unstably.

However, by decreasing the learning rate parameter, a continuously decreasing loss can be obtained. The AUC values for several selected learning rates are shown in Figure 4, with 0.001 achieving the highest AUC value (and being efficient at the same time) and thus also used in the following.

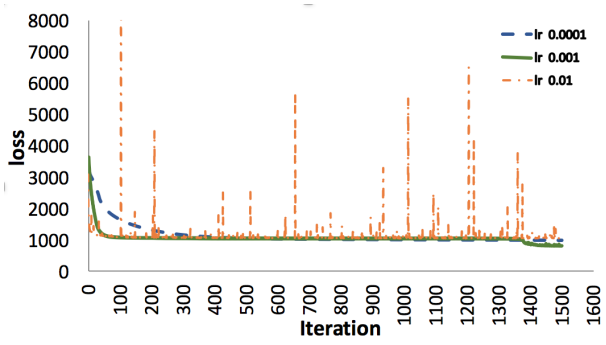

Figure 3: Evaluation of Learning rate versus convergence

learning rate	auc
0.0001	0.929
0.001	0.913
0.01	0.899

Figure 4: Analyzing prediction performance in dependency of learning rate

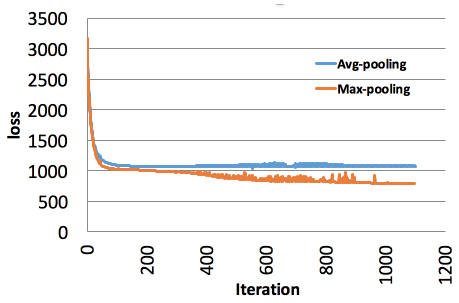

Figure 5: Evaluation of pooling functions

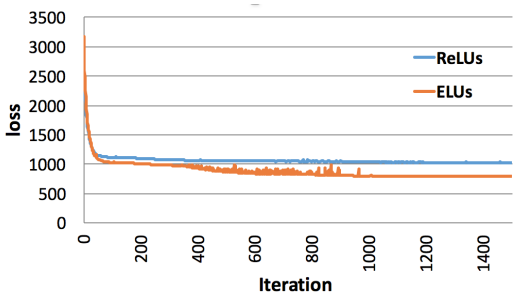

Figure 6: Evaluation of activation functions

Pooling and activation functions. In Figure 5 and 6, we explore the impact of the two pooling functions max-pooling vs. average-pooling on AUC values as well as the activation functions ELUs vs. ReLUs. In Figure 5, it is obvious that by using max-pooling function a lower loss has been achieved. The AUC results in Table 2 confirm that this lower loss is reflected in a better AUC values. Important to mention is that the used activation function in both cases was the ELU. Exploring the impact of ReLU on AUC value versus ELU is shown in Figure 6 as well as in Table 2 under the activation row. Again, here we can conclude that ELU combined with max-pooling returns the best result.

	function	loss	auc
activation	ELUs	785.71	0.929
	ReLUs	1024	0.84
pooling	Max	785.71	0.929
	Avg	1065.7	0.895

Table 2: Evaluation of activation and pooling functions.

Overall, our experimental analysis on our real-word dataset has shown that a two layered CNN architecture including an ELU activation function combined with Max pooling leads to the best AUC results.

5.2. Impact of data properties on the confusion matrix

In industry, it is essential for prediction tasks to reach a high precision and recall on the *minority* class – since our main goal is to avoid malfunctions and high costs. Having imbalanced data, such as in our use case, makes the realization of this goal difficult since often false positives (i.e. functional engines will be predicted as defective) are reported.

As discussed in Section 4.2, we use informative mini batches to handle the imbalanced data, thus, realizing the loss presented in Eq. 3. In the following we compare the classification performance when considering or ignoring the aspect of imbalance (effectively setting $\alpha = 1$ in Eq. 3).

Figure 7 (left) shows the results by considering the corresponding recall-precision curves. While the x axis shows the recall value, presenting the proportion of true positive in the set of truly damaged instances, the y axis refers to precision. The red line shows our principle while the blue line the result when ignoring the imbalance. As clearly shown, our informative minibatches lead to a significantly higher precision even for larger recall values.

Considering the red line, we can observe that for high recall values of around 0.75-0.80, we clearly outperform the blue line – considering the imbalance improves the results. In contrast to the original class label distribution of 1:32 we can reach a level of around 1:1

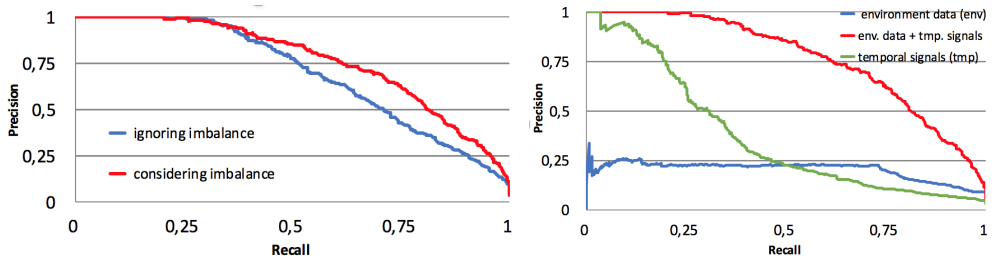

Figure 7: Comparing classification performance on precision-recall curves: left) Prediction when considering class imbalance (red line) clearly outperforms prediction when ignoring class imbalance (blue line). right) Influence of different data sources on prediction performance.

(precison > 0.5) even for high recall values – a significant improvement. *It is also worth noting that our method does not simply lead to a trivial solution where all instances are predicted as being in the majority class* – a common issue for imbalanced data.

Multi-view data. Another important property of our dataset is that the data originates from different sources. In general, we have the two main data sources, one related to the cold test results (temporal data) and the other one related to the (static) environmental data including features such as engines' model or number of cylinders. To examine the relevance of each source on prediction performance, we analyzed each source individually. Figure 7 (right) shows the corresponding recall-precision curve for each dataset.

Considering the lines for the environmental data only, it is clear that the classifier only slightly discriminates the classes. In case of the temporal data, only for small recall values high precision can be reached; followed by a quick drop. However, as it becomes apparent both sources cover different combination: by combined them (our approach; red line) we clearly outperform each individual source. Our proposed architecture realizes this combination in an effective way.

6. Conclusion

In this paper, state of the art convolutional neural networks are used to deal with predicting defective engines from a real-world use case. The data contains multi-view information and is highly imbalanced. To predict the defective engines, we used a two layer CNN-architecture operating on the temporal signal including different attempts on different activation and pooling functions to reach the best results. To deal with the imbalance problem, we used a combination of datalevel and cost sensitive approaches. The experimental results showed that this approach leads to better recall and precision results.

While this paper analyzed for the first time the highly challenging scenario of engine classification based on structure-borne noise measurements there are multiple directions for future work. First, we aim to analyze ensemble methods to further improve the precision and to handle the class imbalance. Additionally, it is of interest to combine our CNN architecture with different sequential models as in e.g. Günnemann et al. (2014a,b) with the goal to capture further dependencies.

References

Pablo Bermejo, Jose A Gámez, and Jose M Puerta. Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Systems with Applications*, 38(3):2072–2080, 2011.

Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Lothar Cremer and Manfred Heckl. *Structure-borne sound: structural vibrations and sound radiation at audio frequencies*. Springer Science & Business Media, 2013.

Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification. *arXiv preprint arXiv:1603.06995*, 2016.

Boubacar Doucoure, Kodjo Agbossou, and Alben Cardenas. Time series prediction using artificial wavelet neural network and multi-resolution analysis: Application to wind speed data. *Renewable Energy*, 92:202–211, 2016.

Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2012.

Nikou Günnemann and Jürgen Pfeffer. Cost matters: A new example-dependent cost-sensitive logistic regression model. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 210–222. Springer, 2017.

Nikou Günnemann, Stephan Günnemann, and Christos Faloutsos. Robust multivariate autoregression for anomaly detection in dynamic product ratings. In *Proceedings of the 23rd international conference on World wide web*, pages 361–372. ACM, 2014a.

Stephan Günnemann, Nikou Günnemann, and Christos Faloutsos. Detecting anomalies in dynamic rating data: A robust probabilistic model for rating evolution. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 841–850. ACM, 2014b.

Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, pages 1725–1732, 2014.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pages 1746–1751, 2014.

Sebastián Maldonado, Richard Weber, and Fazel Famili. Feature selection for high-dimensional class-imbalanced data sets using support vector machines. *Information Sciences*, 286:228–246, 2014.

Maciej A Mazurowski, Piotr A Habas, Jacek M Zurada, Joseph Y Lo, Jay A Baker, and Georgia D Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural networks*, 21(2): 427–436, 2008.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, pages 746–751, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

Jerzy Stefanowski and Szymon Wilk. Selective pre-processing of imbalanced data for improving classification performance. *Lecture Notes in Computer Science*, 5182:283–292, 2008.

Shiliang Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.

Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *IJCAI*, pages 3995–4001, 2015.

Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.