# Experiences with Contrastive Predictive Coding in Industrial Time-Series Classification

Sunanda Gamage
University of Western Ontario
London ON, Canada

Benjamin Klöpper
ABB Corporate Research Center
Ladenburg, Germany

Jagath Samarabandu
University of Western Ontario
London ON

## ABSTRACT

Multivariate time-series classification problems are found in many industrial settings; for example, fault detection in a manufacturing process by monitoring sensors signals. It is difficult to obtain large labeled datasets in these settings, for reasons such as limitations in the automatic recording, the need for expert root-cause analysis, and the very limited access to human experts. Therefore, methods that perform classification in a label efficient manner are useful for building and deploying machine learning models in the industrial setting. In this work, we apply a self-supervised learning method called Contrastive Predictive Coding (CPC) to classification tasks on three industrial multivariate time-series datasets. First, the CPC neural network (CPC base) is trained with a large number of unlabeled time-series data instances. Then, a standard supervised classifier such as a multi-layer perception (MLP) is trained on available labeled data using the output embeddings from the pre-trained CPC base. On all three classification datasets, we see increased label efficiency (ability to reach a goal accuracy level with less labeled examples). In the low data regime (10's or few 100's of labeled examples), the CPC pre-trained model achieves high accuracy with up to 15x less labels than a model trained only on labeled data. We also conduct experiments to evaluate the usefulness of CPC pre-trained classifiers as base models to start an active learning loop, and find that uncertainty sampling does not perform significantly better than random sampling during the initial queries.

## 1. INTRODUCTION

Modern manufacturing plants and process systems are equipped with a large number of sensors that monitor various physical variables on the operation of the systems. These sensors generate vast amounts of time-series data, which can be used for tasks such as quality monitoring, fault detection, and process optimization. Some of these tasks require classification of the generated multi-variate time-series data. Machine learning methods, and especially deep learning methods have shown state of the art performance in these applications [10; 6].

However, deep learning classification methods are typically trained in a fully supervised manner, and they require a large amount of labeled data to achieve high accuracy [13]. In industrial applications, it can be difficult to obtain labeled time-series data due to several reasons [3]. For manual labeling, human experts need to carefully inspect and identify the different classes, which is time-consuming, expensive and also error-prone. Sensor data and labels can be highly process-specific or plant-specific, which makes it impossible to reuse a set of labeled data to train models for multiple plants. Furthermore, some tasks naturally give rise to high class imbalance, and obtaining and labeling data instances from the rare classes is difficult (e.g. faulty class in plant fault detection, because plants do not break down often).

Due to the difficulty in obtaining large labeled datasets in industrial settings, developing time-series classification methods that can be trained in a label-efficient manner to achieve high accuracy is an important research problem. Various approaches have been proposed to address the problem of unavailability of labeled data when training classification models. One solution is to use active learning [18; 4], which interactively queries an expert to label unlabeled instances that are maximally informative to model training. Several forms of semi-supervised learning have also been developed [27; 16], which generally involves the use of both labeled and unlabeled data together to train a mode.

A more recent method designed to improve label efficiency of deep neural network classifiers is self-supervised learning. In self-supervised learning (SSL), a neural network is first pre-trained with a proxy task on unlabeled data, followed by a supervised fine-tuning step on labeled data for the actual classification task. Self-supervised learning has achieved state-of-the art label efficiency (high accuracy with limited labeled data) in many domains such as image [7] and text [24] classification.

In this work, we adapt Contrastive Predictive Coding, a self-supervised learning method introduced in [17] to the problem of label-efficient multivariate time-series classification in industrial applications. The major contributions of this paper are as follows:

- We propose a self-supervised learning pipeline (Fig. 1) that uses Contrastive Predictive Coding (CPC) [17] to train a neural network classifier for multivariate industrial time-series problems in a highly label efficient manner.
- We provide a thorough analysis of the effect of CPC pre-training on label efficiency in three industrial time-series classification datasets, including the very low data regime (10's or few 100's of labeled examples). The three datasets are the Tennessee Eastman Process dataset [22], a proprietary real-world batch process dataset, and a simulated batch process dataset

[26] [1].

- Since CPC models achieve high accuracy with very few training examples, we conduct further experiments to explore the possibility of using the CPC pre-trained method to improve the performance of an active learning loop. Our finding is that with CPC pre-trained models, uncertainty sampling does not have a clear advantage over random sampling in the initial queries of an active learning loop.

The rest of the paper is organized as follows. Section 2 briefly discusses the relevant research in self-supervised learning and time-series classification. Section 3 provides an overview of the Contrastive Predictive Coding method, and section 4 describes the industrial time-series datasets and the classification problems. Section 5 explains the proposed self-supervised learning based classification pipeline. Section 6 presents the details of the experiments and the results analysis. Section 7 provides concluding remarks.

# 2. RELATED WORK

The need to learn efficiently from limited number of labels or obtaining label information in an efficient fashion is not limited to industrial applications and several approaches to address this problem exist.

## 2.1 Semi-supervised learning

Semi-supervised learning tries to leverage unlabeled data in the training process based on two assumptions: cluster assumption and manifold assumptions [29]. The first assumption assumes that the data has cluster structure and instances falling into the same clusters have the same class label. The latter assumes that the data lies on a manifold and nearby data points have similar predictions. Unlabeled data gives insights into which data is similar. Semi-supervised learning has been applied to time-series data as well [27; 16]. One challenge with semi-supervised learning is the possible degradation of performance when using unlabeled data, which leads to the fact that semi-supervised learning is not consistently outperforming supervised learning methods [25].

## 2.2 Self-supervised learning

Self-supervised learning is a machine learning process using deep neural that learns from data using labels that can be obtained in a "semi-automatic" process or that tries to predict parts of the data from other parts [12]. The network trained in a self-supervised fashion is then adapted in a supervised training process to solve a more relevant task. Many studies show, that such a self-supervised pre-training leads to a higher label-efficiency. SSL has also been successfully applied to time-series classification problems, such as phoneme and speaker identification in audio signals [17], and biomedical signal classification [15]. A recent parallel work [19; 20] proposes a self-supervised learning method for time-series classification with a benchmark on an industrial time-series dataset. However, the literature on self-supervised learning methods for industrial multivariate time-series classification problems is lacking.

[1]The simulation dataset is currently in the process of being published.

## 2.3 Active learning

Active learning tries to learn with better performance with fewer labeled samples by enabling the machine learning algorithm to decide from which data it learns. For this purpose, an active learning queries for labels for previously unlabeled samples [23]. The problem of *pool-based active learning* [23], where the active learner chooses samples from a large pool of unlabeled data matches the scenario described in this paper best. Pool-based active learning has previously successfully applied on process data in the chemical industry [21; 4] and other time-series datasets [18]. The performance of active learning depends strongly on the selection of the initial training dataset [8] and the selection of good hyper-parameters during the active learning process [5]. The combination of active learning and self-supervised learning should help to overcome both issues by making the active learning process more robust to the selection of the initial dataset and by using a relatively simple model for the actual classification task.

## 2.4 Combining SSL and Active Learning

The idea to combine self-supervised learning and active learning is not new. For instance, in [11] use learning to rank images w.r.t to *image quality* as a proxy task. Self-supervised learning can also be an approach to overcome the *cold-start* problem of active learning, the issue that the model is not stable when trained with little data at the beginning of the process [28]. The authors applied SSL to pre-train NLP models and use the loss of the pre-trained model as a measure of informativeness in early iterations of the the active learning process [28]. More recent results show that the possible gain of combining SSL and active learning can be limited. In [1] a combination of SSL and random sampling performs better in the low data regime and active learning based sampling is only beneficial in higher-data regimes. The results of [2] even imply that active learning only offers marginal improvements and SSL alone is sufficient to achieve labeling efficiency.

# 3. CONTRASTIVE PREDICTIVE CODING

The self-supervised learning approach to building a neural network classifier involves two steps. First, a neural network is trained on a suitable pre-training task on a large number of unlabeled training examples. This is the representation learning step. Typical pre-training tasks include predicting a masked part of a signal given the remaining parts (e.g. predicting the future given past segments of the signal). This step does not require ground truth labels, which enables the use of any amount of unlabeled data instances available. In the second step (fine-tuning), a classifier (e.g. a fully connected neural network) is trained in a supervised manner with available labeled training examples. In this step, one can choose to either freeze the representation layers of the neural network (no weight updates to the base), or allow weight updates to the base. Designing self-supervised learning methods involves identifying a neural network architecture and a pre-training objective that learns good representations of the data modality. For the multivariate industrial time-series datasets we use in this work, we adapt Contrastive Predictive Coding or CPC [17], a generic self-supervised learning method designed to work with many types of data (images, audio etc.)

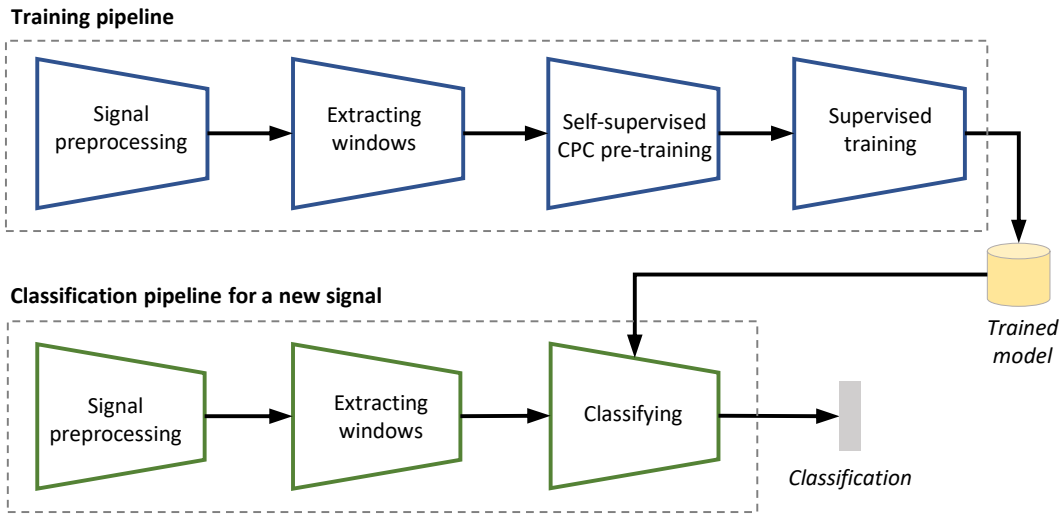


Figure 1: Training and testing pipeline for time-series classification with self-supervised learning.

### 3.1 CPC architecture and self-supervision task

The CPC neural network architecture is illustrated in Fig. 2. We call it the *CPC_Full_SSL_Model*. In the self-supervised pre-training stage, the CPC model is given a sequence of past windows of the time-series, and a second set of future windows (or samples) equal to the number of future steps. However, only one of these samples correspond to the true window of that future step (positive sample), while the others are taken from elsewhere in the dataset (negative samples). The pre-training training objective is to identify the positive sample in the given set of samples, which is formulated as a classification task.

The components of the *CPC_Full_SSL_Model* are described below.

- *A sequence of encoder neural networks.* The encoders have shared weights, and adjacent windows of the time-series ($x_t$) are given as input to each encoder. Each encoder produces a latent vector representation $z_t$. The architecture of the encoder is chosen based on the data modality. For multivariate time-series data, we use a strided 1-D convolutional network.
- *An autoregressive (AR) model.* Typically, this is a recurrent neural network (e.g. GRU or LSTM units). The sequence of encoder outputs is given as input to the AR model. The final output state of the AR model is considered as a single vector representation of the window sequence of the original time-series. It is named the context $c_t$.
- *A set of transformation layers (or networks).* These layers transform the context $c_t$ into the encoder output latent space. The output $\hat{z}_t$ of each layer can be thought of as a latent space prediction of the future window corresponding to the time step. Note that the layers of different time steps have different weights (a unique layer for each time step). We use fully connected layers with linear activations in this work.
- *Dot product and softmax.* These operations output the classification probabilities softmax vector. The element with the highest value is treated as the correct future window (positive sample) among the given set of future windows. There are no trainable weights in these operations.

### 3.2 CPC classifier

Once the CPC model architecture is trained for the self-supervised task of correct future window identification on unlabeled data, we can use the trained weights of part of the model for the supervised classification task. The output of this part of the model is called an embedding, and it is used as input to a classifier for the supervised classification task. We have two options for the choice of embedding.

- Take the encoder and autoregressive (AR) model together and use the final context of the AR model as the embedding (input to train a classifier). This requires a longer context of the input signal (a sequence of windows).
- Use the encoder output (encoded vector) of a single window as input to train a classifier.

In this work, we use the first method, as industrial time-series typically come from long running processes, and our experiments gave better results in this setting. The classifier that is trained on the embeddings is a small fully connected neural network with two hidden layers. We call the model that is trained for the classification the *CPC_Classifier* (encoders + AR model + small fully connected neural network). During the supervised training stage, weight updates are allowed on the entire *CPC_Classifier* network, which includes the base part (encoder and AR model), so that a fine-tuning of the weights occurs.

## 4. INDUSTRIAL DATASETS AND CLASSIFICATION PROBLEMS

To test the suitability of CPC for solving industrial time-series classification tasks with increased label-efficiency, we

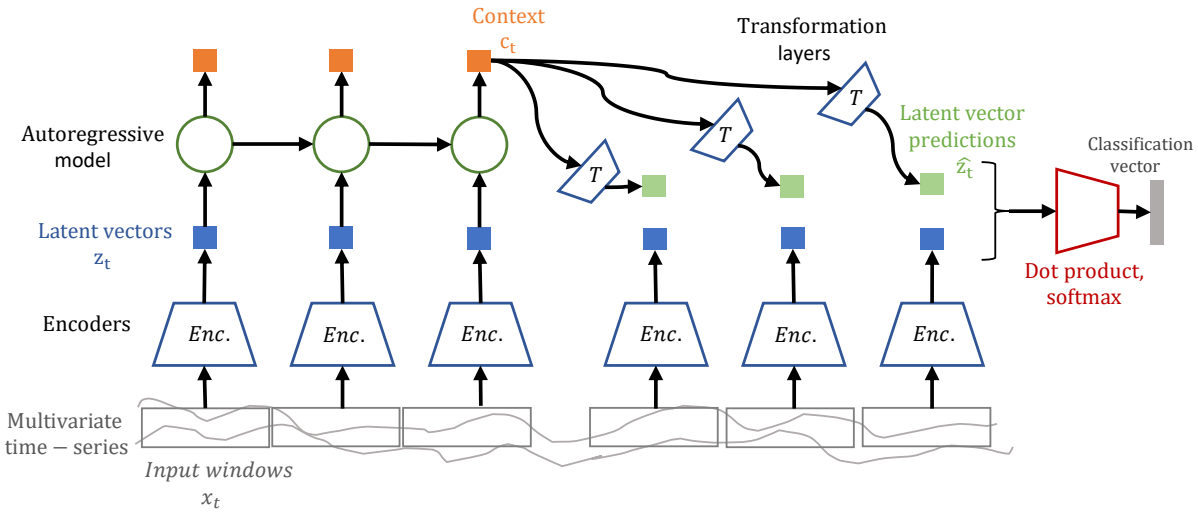


Figure 2: Contrastive predictive coding neural architecture or the *CPC_Full_SSL_Model* (adapted from [17])

applied it to three datasets with associated classification problems. Table 1 shows the important characteristics of these datasets.

TEP refers to the Tennessee Eastman Process, a process simulator that replicates a complex industrial process operated by the Eastman Chemical Company containing five units like reactor, condenser, compressor, separator, and stripper. The simulator and generated datasets are widely used in control research. TEP is a representative of continuous production process, where a continuous flow of input material is transformed into the output material with a fixed process behaviour over a long period of time (in some cases, years). For our experiment we used a publicly available dataset created for the purpose of fault and anomaly detection [22]. The dataset contains 52 time-series where 41 variables represent sensor measurements and 11 are manipulated variables. The variables capture quantities like flow rates, pressures, temperatures, levels, and compressor power output. The public dataset contains 20 different fault scenarios by introducing different types of disturbances. This results in 21 different classes including the absence of disturbances. Besides the presence or absence of disturbances, the different simulation runs differ in the measurement noise. From a practitioners perspective, TEP is an highly idealized benchmark dataset due to its size, well balanced classes and very homogeneous production conditions.

The other two datasets are extracted from batch production processes, which consist of a sequence of steps like heating, cooling, chemical reactions, or stirring. Due to their event driven character, production runs (batches) for the same product can vary considerably. For instance, differences of several hours of batch duration are common [9]. The analysis of batch processes relies on reliable information about the start and end of the production steps and transition are often triggered manually by operators and not consistently documented. Fig. 3 shows the behavior of the level of the unit reactor from the simulated dataset. The colored areas indicate the duration of the different steps or batch phases in the production process. These batch phases are the classes of the time-series classification problem. A related classification problem, to recognize the presence of certain production steps in the data is described in [14]. The different sequential steps of the production take places in so called units and the focus of both batch datasets is a single unit resulting in much smaller number of signals compared to the TEP dataset. The first dataset ("Batch Real") was provided by a provided by a partner of a collaborative research project [2], and it contains data from a real world industrial process. The second dataset ("Batch Simulated") was produced using a process simulator [26] [1].

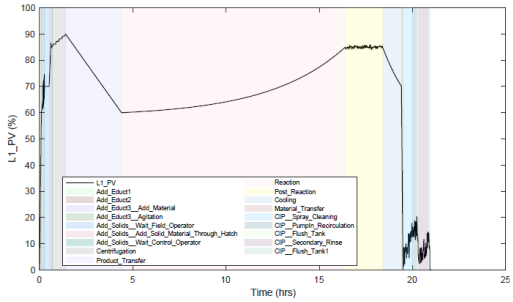


Figure 3: Nominal behaviour of a batch in the simulated dataset. Each highlighted segment belongs to a different batch phase (a class in the classification problem) [26]

## 5. PROPOSED CLASSIFICATION PIPELINE

In order to leverage contrastive predictive coding to train classifiers for multivariate industrial time-series datasets in a label efficient manner, we propose the classification pipeline in Fig. 1. The steps involved are described below.

- *Train-test split.* The individual time-series in the two proprietary batch process datasets (Batch Real and

[2]http://keen-plattform.de/keen/en/

Table 1: Dataset characteristics

| | **TEP** | **Batch Real** | **Batch Simulated** |
|---|---|---|---|
| Data source | Simulation | Real-world batch process | Simulation of batch process |
| Availability | Public [22] | Proprietary [2] | Proprietary [1] |
| Classification task | Fault detection | Batch phase classification | Batch phase classification |
| No. of signals | 52 | 8 | 5 |
| No. of classes | 21 (all taken) | 15 (only 4 taken) | 15 (only 3 taken) |
| Training set size | 21k examples | 6k examples | 4k examples |

Batch Simulated) are split into train and test sets (80%-20% split). The splitting is done in the very first step to avoid any data leaks in later stages. TEP dataset contains a separate test set, therefore splitting is not necessary. From the training sets, a 20% validation set is separated to measure out-of-sample loss during model training and perform early stopping to avoid overfitting.

- *Data preprocessing.* Standard scaling is performed on the individual time-series of the TEP dataset. The time-series in the two batch process datasets were first mean-resampled to obtain mean values per minute. This is necessary, as the different variables in the multivariate time-series come from sensors that are sampled at different sampling intervals. Then, min-max scaling is performed on the individual time-series followed by a data imputing step (linear interpolation) to fill any missing values.

- *Data instance preparation.* The input to the *CPC_Full_SSL_Model* is a sequence of past windows and a set of windows for the future steps with only one correct (positive) window. A sliding window along the time-series is used to prepare these instances. The incorrect (negative) windows for the future steps are taken randomly from anywhere in the dataset. Window size for the TEP dataset is 20, and on the batch process datasets the window size is 10. To increase the number of training examples, we set a 80% overlap of the time-series between two input sequences. The input to the *CPC_Classifier* is a set of past windows prepared in the same manner.

- *Self-supervised pre-training.* The *CPC_Full_SSL_Model* has a strided 1-D convolutional network (no max-pool layers) as the encoder with 5 layers. Each layer has 256 nodes with ReLU activations, kernel sizes: [10, 8, 4, 4, 4] and strides: [5, 4, 2, 2, 2]. This produces a latent space encoded vector of size 256. The autoregressive model is a single-layer recurrent neural network with 128 GRU cells. The AR model sequence length is 10 for the TEP dataset, and 5 for the batch process datasets. The transformation layers are single-layer fully connected networks with 256 linear nodes (to match the encoder output size). The number of future steps is 20 for the TEP dataset, and 5 for the batch process datasets. The pre-training task is the correct future window classification task as described in section 3 on the entire unlabeled training set with cross entropy loss function. We train the network for 80 epochs using the Adam optimizer with learning rate 0.001 and batch size 64.

- *Supervised classifier training.* Once the *CPC_Full_SSL_Model* is pre-trained with unlabeled data, the *CPC_Classifier* model is formed by taking the encoder and AR model and setting a two-layer fully connected neural network on top of it. The two layers have 128 and 64 nodes with ReLU activations with a dropout rate of 0.2. The *CPC_Classifier* is trained for the downstream classification tasks of the datasets (fault detection for TEP dataset and batch phase classification for the batch process datasets) with only labeled data in a supervised manner with cross entropy loss function. Weight updates occur in the full network, which means that the encoder and AR model weights are fine-tuned for the downstream task. We train the network for 100 epochs using the Adam optimizer with learning rate 0.001 and batch size 64.

The hyperparameters of the CPC network architecture are based on the parameters of the original CPC paper [17]. We find that these parameters work well for all three datasets, indicating a degree of architectural robustness that makes the CPC network suitable for different time-series tasks. Parameters such as encoder input window size and autoregressive model sequence length were determined by trial-and-error (trying out a range from small to large).

# 6. EXPERIMENTAL RESULTS

## 6.1 Initial experiments

In order to establish the classification accuracy for each task, we train multiple models with the full labeled datasets. The *CPC_Full_SSL_Model* is first trained for the self-supervision task (correct future window classification) with unlabeled data. For this task, the models give 63% accuracy on the TEP dataset and 86% on the batch process datasets. Fig. 4 shows UMAP visualizations of the CPC embeddings on the test datasets. The TEP test set embedding UMAP shows certain classes forming separate clusters while some classes being scattered in the space. The batch real test set embeddings have well-separated clusters. This observation shows that the CPC embeddings are discriminative for the downstream classification task and provides an indication that the CPC model was well trained.

Next, the *CPC_Classifier* model was trained for the downstream classification tasks in a supervised manner (cross-entropy loss function) with all labeled data in the training sets. Training of the *CPC_Classifier* was done under three

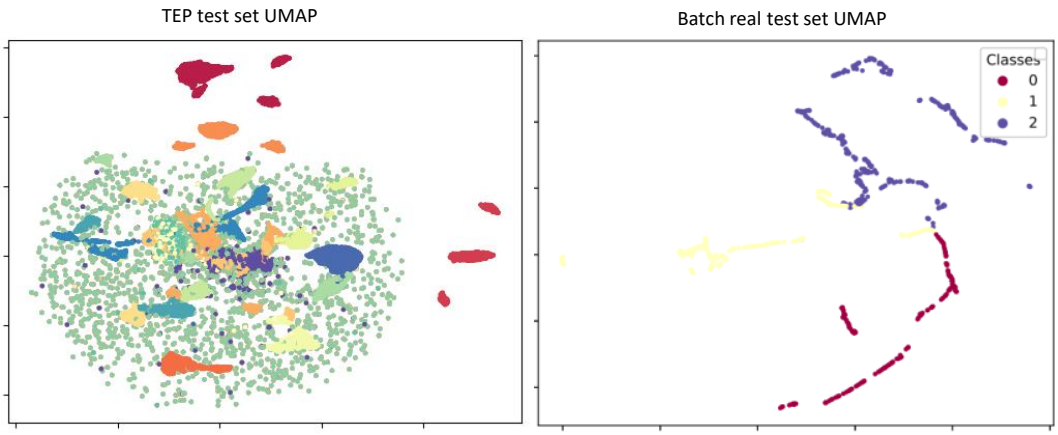


Figure 4: UMAP visualization of CPC embeddings on the test sets of TEP (left) and batch process real (right) datasets. Different colors represent classes of each dataset.

settings, and an LSTM model was also trained. Following is a summary of the training settings. Note that the CPC base refers to the set of convolutional encoders and the autoregressive (GRU) network.

- *LSTM model.* The LSTM network has two hidden layers (128 and 64 nodes) with tanh activations. The length of an input sequence to the LSTM model is equal to the number of samples in an input sequence of the CPC model (200 in the TEP dataset, 50 in the batch process datasets).
- *CPC: no pre-training.* All *CPC_Classifier* weights (1-D CNN encoder, GRU network, fully connected neural network classifier) are randomly initialized and weight updates occur in all components.
- *CPC: pre-trained, base frozen.* Weights of the 1-D CNN encoder and GRU network are pre-trained and frozen, weights of the fully connected neural network classifier are randomly initialized, and weight updates occur only in the fully connected neural network classifier.
- *CPC: pre-trained, base fine-tuned.* Weights of the 1-D CNN encoder and GRU network are pre-trained, fully connected neural network classifier weights are randomly initialized, and weight updates occur in all components.

The results are summarized in Table 2. On the TEP multi-class classification problem (with all 21 classes), the training settings where all network weights are updated yield accuracy upward of 90%. The classifier trained on CPC embeddings (*CPC: pre-trained, base frozen* setting) yields a relatively lower accuracy of 82%. This is likely due to the fact that the representations learned by the CPC base from the self-supervision task needs fine-tuning in the downstream task of fault detection for higher performance. The LSTM network also achieves a comparable 81.5% accuracy. The accuracy figures on the TEP dataset are comparable with the 81.43% accuracy reported in [19]. However, a direct comparison is difficult due to differences in how the data instances are prepared (different windowing techniques, subset of classes used etc).

On the batch process datasets, all CPC model training settings achieve near perfect classification. In addition to establishing a comparison of the possible training methods, this experiment also verifies that the neural network architecture of the *CPC_Classifier* is indeed capable of learning a classification function on the three datasets given all available labeled data. On all three datasets, the CPC models perform significantly better than the LSTM network.

## 6.2 Label efficiency experiments

To benchmark the label efficiency of the contrastive predictive coding pipeline, we conducted the four types of model training described in the previous section (Table 2). In the downstream task supervised training stage, the models were trained with subsets of the full labeled training sets. In order to observe the effects of CPC pre-training on the very-low-data regime, we trained models with as little as 0.25% of the full training set, which corresponds to 10 - 40 examples in the three datasets. Such minimal availability of labeled data is possible in certain industrial applications. The resulting label efficiency curves for the three datasets are shown in Fig. 5.

On all three datasets, the self-supervised pre-trained *CPC_Classifier* gives high accuracy compared to the randomly initialized *CPC_Classifier* and baseline LSTM model when trained with as little as 0.25% of the full training set (e.g. 72.8% accuracy vs. 35.8% accuracy on the TEP dataset with 0.25% labeled training examples). Note that this very-low-data regime consists of 42 labeled training examples in the TEP dataset and 10 labeled training examples in the batch process real dataset.

The large label efficiency gain of the pre-trained CPC models persists further in to the low-to-mid-data regime (up until 20% or more of labeled training data is made available). On the TEP dataset, the pre-trained CPC models achieve 78% accuracy with about 10x fewer labels, and 82% with about 2.5x fewer labels than the randomly initialized CPC model or the LSTM network. On the batch process real dataset, the pre-trained CPC models achieve 98% accuracy with 100x fewer labels than the randomly initialized model or the LSTM network. A similar large efficiency gain is seen on the batch process simulated dataset (100% accuracy with

Table 2: Classification accuracy on the three datasets (trained on all labeled data).

| **Training method** | **TEP** | **Batch real** | **Batch simulated** |
|---|---|---|---|
| LSTM model | 81.5% | 95% | 85.6% |
| CPC with no pre-training | 90.12% | 99.33% | 98.6% |
| Pre-trained + frozen CPC base | 82% | 99.7% | 99.3% |
| Pre-trained + fine-tuned CPC base | 92.3% | 99.4% | 100% |

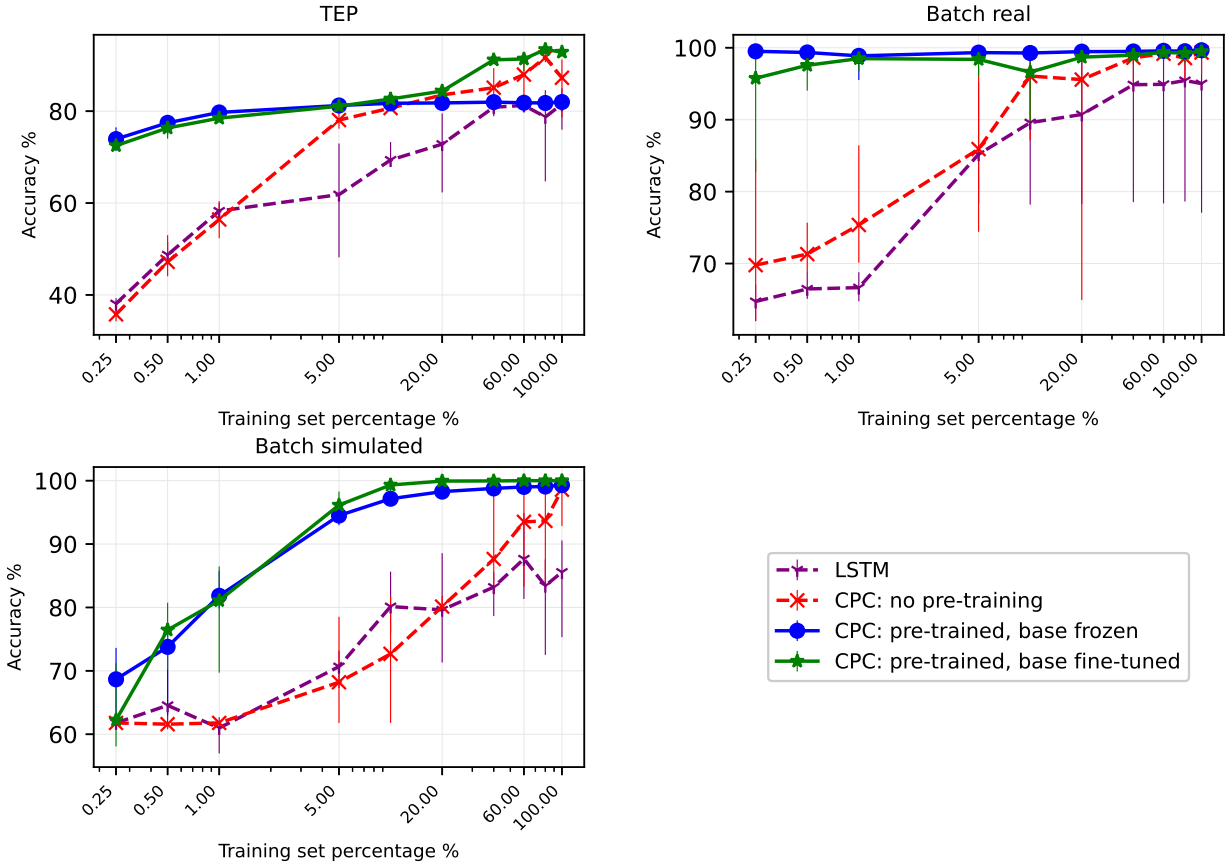


Figure 5: Label efficiency curves of different types of *CPC_Classifier* training on the three datasets. Top left: TEP dataset, top right: batch process real dataset, bottom: batch process simulated dataset. Vertical error bars indicate minimum and maximum values in experiment repetitions.

10x fewer labels). All models converge to roughly the same accuracy values when they are trained with more than 20% of labeled training examples.

Small labeled training sets can be the realistic case in certain industrial time-series classification problems, and our results indicate that a pre-trained CPC model is beneficial in these applications.

## 6.3 Active learning experiments

Active learning algorithms attempt to reduce human labeling effort by choosing most informative or important unlabeled examples and prompting a human expert to label them. Typically, a model is first trained on available labeled data, and its output is then used to select unlabeled examples to be labeled by a human expert. In the case of uncertainty sampling, examples where the model is most uncertain (e.g. largest softmax value is small) are selected. When the initial model is not well-trained, its output is unreliable, which makes it difficult for an a active learning algorithm to select good examples.

Since the self-supervised pre-trained *CPC_Classifier* gives high accuracy even when trained with a small number of labeled examples, we use it as the initial model in an active learning loop instead of a randomly initialized model. The model is trained for the classification task on an initial labeled subset (labeling budget) of 0.1% of the full training set. Then, batches of unlabeled examples are selected via uncertainty sampling and random sampling (batch size: 20 on TEP dataset, 10 on the batch process real dataset). The labels of these examples are then revealed (as if they were labeled by a human expert), and the model is trained with the set of all available labeled examples (initial labeled subset and the set of examples whose labels were revealed via the active learning loop).

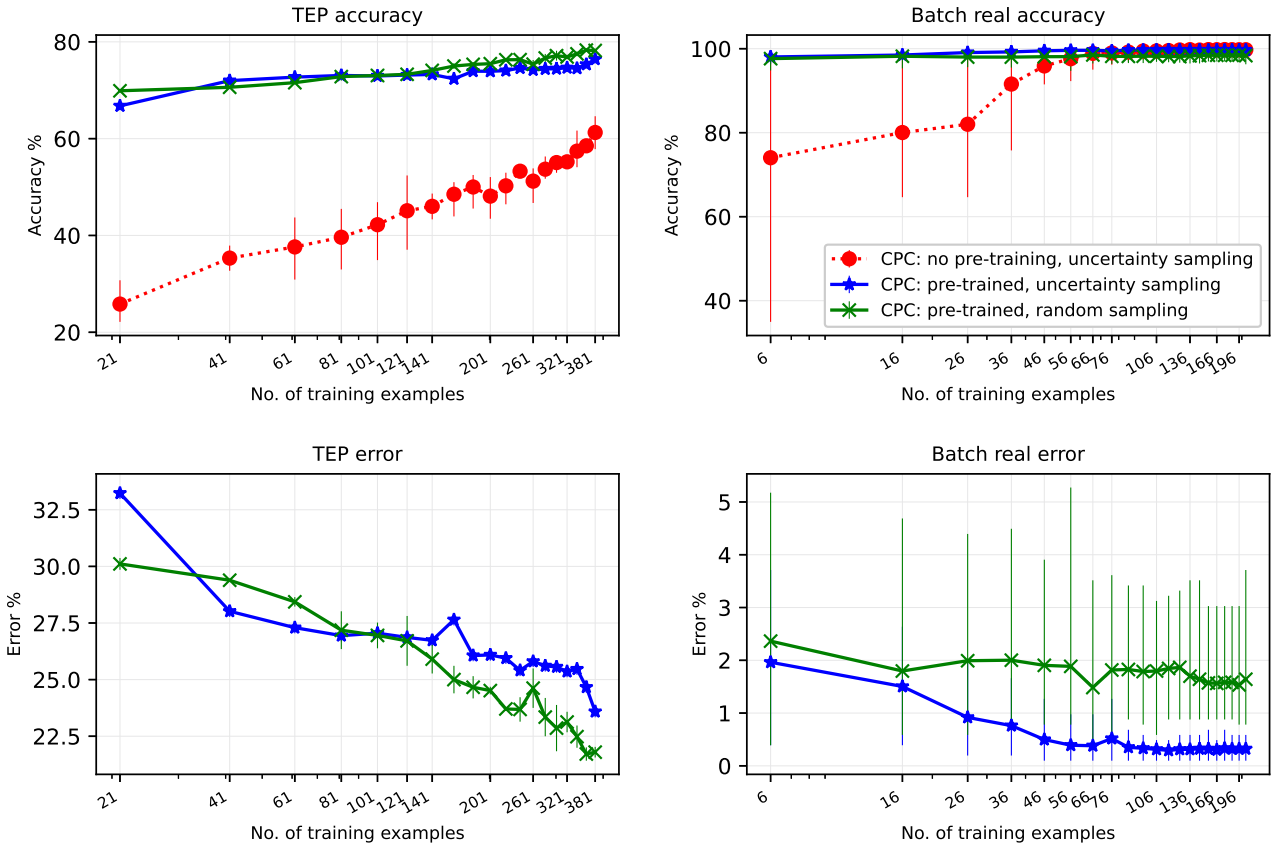


Figure 6: Active learning curves (accuracy and error) of on two datasets. Left: TEP dataset, right: batch process real dataset. Vertical error bars indicate minimum and maximum values in experiment repetitions.

The resulting active learning performance curves are shown in Fig. 6. On the TEP dataset, the self-supervised pre-trained *CPC_Classifier* does not provide a clear advantage to uncertainty sampling over random sampling. On the Batch real dataset, uncertainty sampling of the pre-trained model produces a lower error than random sampling as more queries are made. However, the maximum difference of average error between the two sampling methods is around 1.57%, which may not be considered as a significant gain by uncertainty sampling, especially during the initial few queries. On both datasets, the self-supervised pre-trained *CPC_Classifier* models start out with a much higher accuracy than the CPC models with no pre-training (randomly initialized), and the gap continues through the low-data regime. This observation is consistent with the results of the label efficiency experiments (section 6.2, Fig. 5).

While there is no prior work that combines self-supervised and active learning on the TEP dataset for direct comparison, such techniques applied to image data have produced similar results. In [1], authors report that random sampling with a self-supervised pre-trained CNN model on multiple image datasets performs better than active learning sampling methods based on entropy and embeddings distance, especially in the low-data regime. Similarly, authors of [2] find that active learning algorithms provide no additional benefit for image classification when combined with self-supervised and semi-supervised methods.

These results indicate that active learning with uncertainty sampling may not be a worthwhile time investment when building neural networks for industrial time-series classification problems with small labeled datasets. However, if a practitioner plans to use active learning to minimize labeling effort, self-supervised pre-training of the models is a good first consideration as the pre-trained models perform better in the low-data regime.

## 7. CONCLUSION

In this paper, we presented a self-supervised learning pipeline based on contrastive predictive coding or CPC for multivariate industrial time-series classification in a label efficient manner. This method leverages large amounts of available unlabeled time-series data for pre-training a CPC neural network to learn valid representations for downstream classification tasks. We examined the label efficiency gains of the proposed pipeline on three time-series classification datasets. In all cases, the CPC pre-trained models gave high accuracy with a very small number of labeled examples.

We also explored the possibility of using a CPC-pre-trained model to cold-start an active learning loop. We found that a loop with uncertainty sampling does not perform better than random sampling during the initial queries on the time-

series datasets.
In many industrial applications with classification problems, vast amounts of unlabeled time-series data is available, yet the human expertise to label them is expensive or unavailable. In comparison, the computational cost of self-supervised pre-training is negligible. Therefore, the proposed CPC pipeline is an excellent candidate solution for such industrial classification problems.
In future work, we will extend this project to a benchmark to evaluate more self-supervised methods in multivariate industrial time-series applications. It is important to evaluate these methods in the presence of typical data related issues such as heavy class imbalance (e.g. anomaly detection; much fewer failure cases than normal operation), varying noise levels due to uncalibrated sensors, missing values, and out-of-distribution test-time settings (e.g. due to varying raw material qualities or differing operational conditions).

# 8. REFERENCES


[1] J. Z. Bengar, J. van de Weijer, B. Twardowski, and B. Raducanu. Reducing label effort: Self-supervised meets active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1631–1639, 2021.

[2] Y.-C. Chan, M. Li, and S. Oymak. On the marginal benefit of active learning: Does self-supervision eat its cake? In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3455–3459. IEEE, 2021.

[3] M. Gärtler, V. Khaydarov, B. Klöpper, and L. Urbas. The machine learning life cycle in chemical operations–status and open challenges. *Chemie Ingenieur Technik*, 93(12):2063–2080, 2021.

[4] Z. Ge. Active learning strategy for smart soft sensor development under a small number of labeled data samples. *Journal of Process Control*, 24(9):1454–1461, 2014.

[5] Y. Geifman and R. El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

[6] A. Gupta, H. P. Gupta, B. Biswas, and T. Dutta. Approaches and applications of early classification of time series: A review. *IEEE Transactions on Artificial Intelligence*, 1(1):47–61, 2020.

[7] O. Henaff. Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning*, pages 4182–4192. PMLR, 2020.

[8] R. Hu, B. Mac Namee, and S. J. Delany. Off to a good start: Using clustering to select the initial training set in active learning. In *Twenty-Third International FLAIRS Conference*, 2010.

[9] S. Lee, T. O'Connor, X. Yang, C. Cruz, S. Chatterjee, R. Madurawe, C. Moore, L. Yu, and J. Woodcock. Modernizing pharmaceutical manufacturing: from batch to continuous production. *Journal of Pharmaceutical Innovation*, 10, 03 2015.

[10] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi. Applications of machine learning to machine fault diagnosis: A review and roadmap. *Mechanical Systems and Signal Processing*, 138:106587, 2020.

[11] X. Liu, J. Van De Weijer, and A. D. Bagdanov. Exploiting unlabeled data in cnns by self-supervised learning to rank. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1862–1878, 2019.

[12] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[13] G. Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

[14] S. Merkelbach, R. Tan, F. Bähner, M. Gärtler, J. Gatter, R. Gedda, and L. Urbas. Automatic detection of start and end times of batch phases for the process industry. In *Automation 2021*, Automation, Baden-Baden, DE, 2021. VDI.

[15] M. N. Mohsenvand, M. R. Izadi, and P. Maes. Contrastive representation learning for electroencephalogram classification. In *Machine Learning for Health*, pages 238–253. PMLR, 2020.

[16] M. N. Nguyen, X.-L. Li, and S.-K. Ng. Positive unlabeled learning for time series classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. AAAI Press, 2011.

[17] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[18] F. Peng, Q. Luo, and L. M. Ni. Acts: an active learning method for time series classification. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 175–178. IEEE, 2017.

[19] J. Pöppelbaum, G. S. Chadha, and A. Schwung. Contrastive learning based self-supervised time-series analysis. *Applied Soft Computing*, page 108397, 2022.

[20] T. Pranavan, T. Sim, A. Ambikapathi, and S. Ramasamy. Contrastive predictive coding for anomaly detection in multi-variate time series data. *arXiv preprint arXiv:2202.03639*, 2022.

[21] G. K. Raju and C. L. Cooney. Active learning from process data. *AIChE journal*, 44(10):2199–2211, 1998.

[22] C. A. Rieth, B. D. Amsel, R. Tran, and M. B. Cook. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation, 2017.

[23] B. Settles. Active learning literature survey. 2009.

[24] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification? In *China national conference on Chinese computational linguistics*, pages 194–206. Springer, 2019.

[25] J. E. Van Engelen and H. H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[26] M. Vicente. Model for batch process data generation - a benchmark for machine learning testing and comparison. page 118, 09 2021.

[27] L. Wei and E. Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2006. Association for Computing Machinery.

[28] M. Yuan, H.-T. Lin, and J. Boyd-Graber. Cold-start active learning through self-supervised language modeling. *arXiv preprint arXiv:2010.09535*, 2020.

[29] Z.-H. Zhou. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018.