Looking for Change: A Computer Vision Approach for Concept Drift Detection in Process Mining

Alexander Kraus[1] and Han van der Aa[2]

[1] Data and Web Science Group, University of Mannheim, Germany
[2] Faculty of Computer Science, University of Vienna, Austria
alexander.kraus@uni-mannheim.de
han.van.der.aa@univie.ac.at

Abstract. Concept drift in process mining refers to a situation where a process undergoes changes over time, leading to a single event log containing data from multiple process versions. To avoid mixing these versions up during analysis, various techniques have been proposed to detect concept drifts. Yet, the performance of these techniques, especially in situations when event logs involve noise or gradual drifts, is shown to be far from optimal. A possible cause for this is that existing techniques are developed according to algorithmic design decisions, operating on assumptions about how drifts manifest themselves in event logs, which may not always reflect reality. In light of this, we propose a completely different approach, using a deep learning model that we trained to learn to recognize drifts. Our computer vision approach for concept drift detection (`CV4CDD`) uses an image-based representation that visualizes differences in process behavior over time, which enables us to subsequently apply a state-of-the-art object detection model to detect concept drifts. Our experiments reveal that our approach is considerably more accurate and robust than existing techniques, highlighting the promising nature of this new paradigm.

Keywords: Process mining · Concept drift detection · Object detection · Computer vision · Deep learning.

1 Introduction

Business processes are widely supported by information systems, which record data generated during the execution of process instances. Such data, captured in the form of event logs, forms the basis for *process mining*, a family of techniques that analyze how business processes are truly executed [1]. The *event logs* used in process mining represent snapshots of data generated during process execution over a specific period of time. Due to their dynamic environments, business processes under analysis are often not in a steady state but are rather subject to changes [3]. These changes can result in the presence of *concept drifts* in event logs, i.e., situations in which an event log contains data from multiple versions of

a process. To avoid polluting process mining results by mixing up these different versions, *concept drift detection* strives to identify them [3].

The importance of concept drift detection in process mining has been widely acknowledged [24], resulting in a range of proposed concept drift detection techniques [7, 21]. What these techniques have in common is that they are all developed according to certain algorithmic design decisions, relying on heuristic strategies to identify concept drifts. Essentially, these techniques operate based on assumptions about how drifts manifest themselves in event logs. However, such assumptions and their corresponding design decisions may not hold in all settings, which means that the techniques may break down in situations that involve noise, drifts of different types, or change severity. As a result, existing techniques cannot consistently detect concept drifts with proper latency.

To overcome this problem, we propose a computer vision approach for concept drift detection (`CV4CDD`) that follows a completely different paradigm. More precisely, our approach uses a machine learning (ML) model that was trained on a large collection of labeled data, allowing it to learn how drifts actually manifest themselves in event logs. The possibility of training such a model has only recently emerged, thanks to a tool for generating large collections of logs with known concept drifts [8]. Still, even with such labeled data at hand, the question of how to tackle concept drift detection through (supervised) ML is far from straightforward. This is due to the difficulty of capturing the progression of an entire process over time, in a manner suitable as input for an ML model. To deal with this, we draw inspiration from works that use image-based representations to encode multi-faceted data about processes [19, 20]. Specifically, we turn an event log into an image that visualizes differences in process behavior over time. This enables us to employ a state-of-the-art object detection model [13] (from the field of computer vision), fine-tuned on a large collection of event logs with known concept drifts, to recognize where drifts occur in unseen event logs. Our experiments reveal the efficacy of this idea, showing that our approach is considerably more accurate and robust than existing techniques.

In the remainder, we define the scope of the work in Section 2, including the input to the problem and desired output. Our `CV4CDD` approach is detailed in Section 3. Section 4 presents the evaluation of our approach against state-of-the-art techniques. Finally, Section 5 reflects on related work before Section 6 concludes the paper.

2 Problem Definition

This section outlines the concept drift detection problem that our work focuses on. We start by defining the input to the problem, before describing the scope, and desired output.

Problem input. Our approach takes as input an event log L, which we define as a collection of events e recorded by an information system during process execution. Each event $e \in L$ needs to at least have a *case ID*, an *activity*, and a *timestamp*. We use Σ_L to refer to the set of traces, where each $\sigma \in \Sigma_L$ represents

a sequence of events from L with the same case ID, ordered by their timestamps. Traces in Σ_L are ordered by the timestamp of their first event.

Scope. Our work focuses on the most important problem in concept drift detection, i.e., the *detection of process changes* in the control-flow perspective of a process. A *process change* refers to a situation where a business process undergoes a modification that affects the characteristics and behavior of the process, leading to a new *process version*, i.e., a specific variant of a business process. The detection of process changes consists of two sub-tasks [16]:

1. *Change point detection*, which aims to detect *if* and *when* a process change occurred in an event log. This is captured in the form of *process change point* p, a moment in an event log when the process behavior starts to differ significantly from the previous behavior.
2. *Change characterization*, which considers *how* process changes manifest themselves over time. In our work, we focus on the two fundamental kinds of process changes: sudden and gradual drifts, as illustrated in Figure 1. A *sudden drift* is described by an abrupt change in process behavior after a certain process change point, while a *gradual drift* is characterized by a transition period between two change points when a current process version is replaced by a new one, and both versions coexist during a transition period [3].

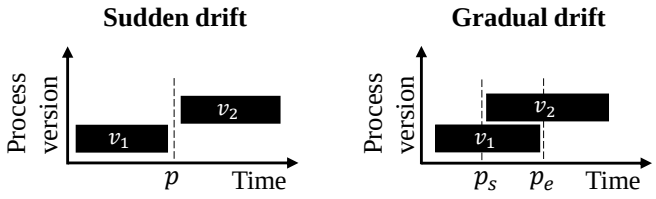

Fig. 1: The scope of our work: detecting sudden and gradual drifts and the corresponding process change points (adapted from [3]).

Output. Given the above, the desired output of the problem we consider is a collection of detected concept drifts $D^d = \{d_1^d, \ldots, d_n^d\}$, where each concept drift is characterized by a drift type $d_i^d.type$ (either sudden or gradual) and one or two change points, i.e., $d_i^d.p$ for sudden drifts or $d_i^d.p_s$ and $d_i^d.p_e$ for gradual ones.

3 Approach

This section introduces `CV4CDD`, our computer vision approach for concept drift detection. As visualized in Figure 2, `CV4CDD` consists of two main steps. First, we transform an event log into an image that captures the behavioral (dis)similarity of a process over time. Then, the image is passed to our fine-tuned computer vision model, which identifies if drifts are present in an event log and, if so, determines their type (sudden or gradual) and corresponding change points.

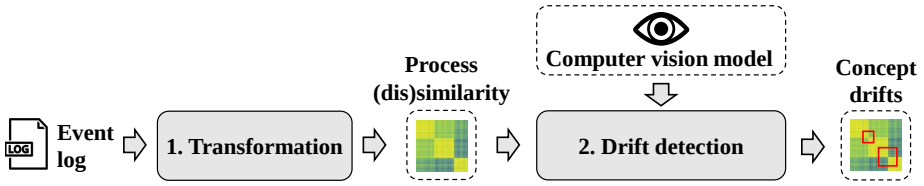

Fig. 2: CV4CDD: overview of the main steps.

3.1 Transformation of Event Log into Image

The first approach step takes as input an event log and transforms it into an image as output. The image visualizes behavior (dis)similarity of a process over time recorded, which can be used to recognize concept drifts. The transformation includes four steps, as depicted in Figure 3.

Fig. 3: First approach step: transforming an event log into an image.

Split traces into windows. The approach first splits the chronologically ordered traces in Σ_L into an ordered collection of N windows, $W := \langle w_1, \ldots, w_N \rangle$. These windows are non-overlapping and collectively cover all recorded traces in Σ_L, with each window w_i containing approximately $|\Sigma_L|/N$ traces.

During the fine-tuning of the computer vision model, we use 200 as a default value for the number of windows N so that each window w_i covers about 0.5% of all traces from the log. For the training collection (see Section 4.1), with logs containing 2,000–8,000 traces, this setting provides an effective representation per log. We recommend using 200 windows also as the default setting for detecting concept drifts using CV4CDD on a new event log. However, for small event logs (e.g., with fewer than 2,000 traces), decreasing the number of windows avoids having too few traces per window. Conversely, larger event logs (especially those spanning a long time range) may benefit from having more windows, to prevent drifts occurring within the span of a single window.

Calculate behavioral representation. After establishing W, the approach computes a *behavioral representation*, $B(w_i)$, for each set of traces in w_i to characterize the recorded process behavior. $B(w_i)$ consists of two-dimensional tuples, each storing a behavioral pattern and its frequency. A common behavioral representation used in process mining is to capture the directly-follows frequencies observed during a time window [7], which we use as the default behavioral representation. It counts how often two activities were observed to directly succeed

each other for a case. However, it is important to note that the choice for a behavioral representation is flexible, provided that it yields a numeric frequency distribution over a predefined set of relations or patterns across the window. Therefore, `CV4CDD` can also cover other types of relations (e.g., eventually follows), sets of relation types, such as those of a behavioral profile [23], or declarative process constraints [6].

Measure similarity. Next, our approach compares the behavioral representations obtained for the different windows, quantifying their similarity. This comparison is done for each pair w_i and w_j from W, resulting in a symmetric similarity matrix denoted as S. Each entry $S[i, j]$ in this matrix shows the similarity between the behavior represented by $B(w_i)$ and $B(w_j)$.

The similarity matrix S can be established using various similarity measures. Common options employed in process mining contexts include the cosine similarity (our default choice) and the Earth mover's distance [4].

Figure 4 illustrates the calculation of the similarity measure between two windows, using a behavioral representation based on directly-follows frequencies and the cosine similarity measure.

Windows	**Traces**	**Behavioral representation**	**Similarity measure**
w_i	$\langle$a,b,c$\rangle^2$ $\langle$a,c$\rangle$	$B(w_i) = \begin{pmatrix} a \to b : 2 \\ b \to c : 2 \\ a \to c : 1 \\ c \to d : 0 \end{pmatrix}$	$S[i,j] = sim(B(w_i), B(w_j))$ $= cosine(\begin{bmatrix}2\\2\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\1\\1\end{bmatrix})$ $= 0.83$
w_j	$\langle$a,b,c$\rangle$ $\langle$a,c,d$\rangle$	$B(w_j) = \begin{pmatrix} a \to b : 1 \\ b \to c : 1 \\ a \to c : 1 \\ c \to d : 1 \end{pmatrix}$	

Fig. 4: Illustration of the similarity calculation between two windows.

Transform into image. Finally, to enable image-based concept drift detection, the similarity matrix S is visualized in the form of an image. Figure 5 depicts a few examples of this transformation, covering different drift scenarios recorded in an event log.

To convert a similarity matrix into an image, our approach normalizes the matrix values to a range between 0 and 1, where the maximum similarity corresponds to 1 and the minimum similarity is 0. Each normalized value is then scaled by 255 and converted to integers, resulting in a range between 0 and 255. Finally, using the Python Imaging Library[3] and a color map, images are generated from the normalized values.

[3] Available online: https://python-pillow.org

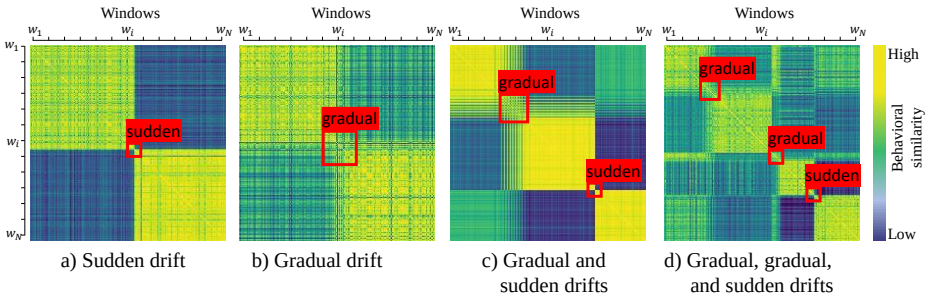

Fig. 5: Output of the first approach step (incl. annotations).

It is important to note that CV4CDD can also work with other types of images that record changes in process behavior over time. For instance, it could use images obtained from the Visual Drift Detection techniques introduced by Yeshchenko at el. [25]. In this case, a new annotation and training procedure for the drift detection model (see Section 3.2) may be necessary, though.

3.2 Drift Detection

The second step of CV4CDD takes as input the image obtained from the previous step and applies a fine-tuned object detection model to detect concept drifts. In this section, we present an overview of the object detection task and employed object detection model, elaborate on the training data and its annotation, and clarify the training configurations.

Object detection using RetinaNet. Object detection is a fundamental task in computer vision, where the goal is to identify and locate objects within images. Deep learning methods have significantly advanced this field by directly learning features from data, leading to breakthroughs in object detection [13]. *RetinaNet* [11] is a recent addition to deep learning-based object detection models. Known for its effectiveness and reliability, RetinaNet has become widely adopted in both research and practical applications, setting new standards in object detection performance.

The RetinaNet model has three main parts [11]. First, a backbone network extracts features from input images, capturing important patterns for identifying objects. Second, a feature pyramid network is added on top of the backbone to create a multi-scale pyramid of feature maps, combining features from different levels of the backbone, enabling effective detection of objects of various sizes. Finally, the model includes two subnetworks for classification and bounding box regression, enabling accurate object detection. As a result, the RetinaNet architecture is well-suited for detecting concept drift in images because it can effectively handle noise and identify sudden and gradual drifts across various sections of an image.

Training data and annotation. We use a training set of event logs with known concept drifts. Each event log in the training set is transformed into an image

using the first step of our CV4CDD approach (Section 3.1). Then, each image is annotated based on the drift information stored in the gold standard, capturing where different drifts occur and what their types are. For the annotation, we define bounding boxes using widely-employed COCO (Common Objects in Context) format [12].

In our case, we use these bounding boxes to capture where drifts of certain types occurred in the image, as shown for various scenarios in Figure 5. To annotate sudden drifts, characterized by a single change point p that belongs to a window w_i, we establish a bounding box around w_i that spans 5 windows in both directions from w_i. For gradual drifts, we create annotations using windows that correspond to the start and end change points. Each change point p_s and p_e is associated with a trace index, which belongs to a particular window in W. The corresponding windows w_i and w_j, allow us to link p_s and p_e to their window indices i and j. We use these indices to define a bounding box for gradual drift within an image.

Training configurations. To operationalize CV4CDD, we specifically use the RetinaNet model from the TensorFlow Model Garden[4], based on the SpineNet backbone (ID 143). The model is pre-trained on the COCO dataset[5], a widely-used dataset for object detection. To adapt it to our specific task, we fine-tune the model on a training set and halt fine-tuning using a validation set to prevent overfitting.

Image input, batch size, anchor boxes. We use a fixed input size of 256×256 for fine-tuning the model[6] with a batch size of 64, taking 312 iterations per epoch. The model undergoes 500 epochs of training, with augmented images to increase diversity and robustness by scaling up to two times or down to one-tenth of their original size. We employ anchor boxes with a 1:1 aspect ratio, concentrating exclusively on square-shaped bounding boxes, since all annotations correspond to squares of different sizes along the diagonal of the image.

Optimization and learning rate. We use stochastic gradient descent with a momentum of 0.9 and clip norm of 10, known for its simplicity and efficiency in training deep learning models, especially with large datasets. Momentum aids convergence by leveraging past gradients, while gradient clipping prevents exploding gradients, ensuring stability, particularly in complex neural networks. Additionally, we employ a cosine learning rate, widely adopted for its simplicity and ability to enhance convergence and generalization. This schedule adjusts the learning rate throughout training, following a cosine-shaped function.

Based on the fine-tuned RetinaNet model, our CD4CDD approach can be directly applied to unseen event logs to detect concept drifts.

[4] TensorFlow Model Garden, Available online: https://github.com/tensorflow/models
[5] COCO dataset, Available online: https://cocodataset.org/
[6] If an input image provided for inference has a different pixel size, i.e., because it was established using a different number of windows N, RetinaNet automatically rescales the image to the default size.

4 Evaluation

This section reports on the experiments we conducted to evaluate the performance of our `CV4CDD` approach for concept drift detection, in particular, also in comparison to a range of baselines. To ensure reproducibility, we have made the data collection, implementation, configurations, and raw results accessible in our public repository[7].

Section 4.1 describes the data that we used, Section 4.2 reports on Experiment 1, focusing on change point detection, and Section 4.3 on Experiment 2, focusing on drift detection.

4.1 Data Collection

Our data collection comprises two datasets, summarized in Table 1.

CDLG dataset. To train, validate, and test our drift detection approach, we require a large collection of event logs that contain known (i.e., gold-standard) concept drifts of sudden and gradual types. Since such a collection is not publicly available, we, therefore, generated synthetic datasets using CDLG (Concept Drift Log Generator) [8], a tool for the automated generation of event logs with concept drifts, which comes with a wide range of parameters.

Table 1: Characteristics of the two datasets.

Number of	**CDLG dataset**			**CDRIFT dataset**
	Training	**Validation**	**Test**	
Event logs	20000	1250	3750	115
→ without drifts	5022	302	873	0
→ with noise	10064	617	1860	60
Change points	44997	1906	8547	156
Drifts	29943	1893	5691	156
→ Sudden drifts	14889	953	2835	156
→ Gradual drifts	15054	940	2856	0

We used CDLG to generate 25,000 event logs, allocating 80% for training, 5% for validation, and 15% testing. Key settings used for their generation:

- The logs are generated from process trees containing between 6 and 20 activities, as well as sequential, choice, parallel, and loop operators.
- Each event log has between 2,000 and 8,000 traces (with an average of around 4,600 traces per log), and the average trace length varies from 1 to 65 events.
- The event logs have 0 to 3 drifts each (with equal probability). Drifts are either sudden or gradual, yielding a maximum of 6 change points per log.

[7] Project repository: https://gitlab.uni-mannheim.de/processanalytics/cv4cdd.

- Each drift introduces changes to about 30% of the process tree elements (activities and operators), through deletion, insertion, or swapping.
- A quarter of the logs contain randomly inserted noise in 30% of the traces and another quarter in 60% of the traces. The other half are noise-free.

CDRIFT dataset. To assess the generalizability of our approach and verify that its performance is not restricted to the characteristics of the CDLG dataset, we also consider a dataset used in a recent experimental study [2], which we refer to as the CDRIFT dataset. This set consists of 115 synthetic event logs, previously employed in evaluating various concept drift detection techniques, stemming from three sources [5, 18, 24]. The logs have about 1700 traces on average and contain between 1 and 3 sudden drifts. Notably, the CDRIFT dataset does not contain any gradual drifts.

4.2 Experiment 1: Change Point Detection

In this section, we present the experiment conducted to evaluate the change point detection performance of our approach in comparison to existing baselines. We isolate this task from change characterization, given its fundamental role in concept drift detection, as also evidenced by the various techniques that have been proposed to address it. In the following, we discuss the evaluation setup and obtained results.

Evaluation Setup. Below we elaborate on the details of the baselines, evaluation measures, as well as configurations used to evaluate our approach.

Baselines. We compare our approach to seven change point detection techniques that were used in a recent benchmark study by Adams et al. [2]:

1. Bose/J by Bose et al. [3] uses non-overlapping and continuous fixed-size windowing with activity pair-based feature extraction and statistical testing.
2. Adwin/J by Martjushev et al. [16] improves the Bose/J technique by introducing adaptive windowing using the ADWIN approach.
3. ProGraphs by Seeliger et al. [22] implements non-overlapping and continuous adaptive-size windowing, uses graph-based process features alongside Heuristics Miner, and employs statistical testing.
4. ProDrift by Maaradji et al. [15] employs non-overlapping continuous fixed and adaptive-size windowing, statistical testing, and an oscillation filter.
5. Rinv by Zheng et al. [26] uses behavioral profiles, a process similarity measure, and DBSCAN clustering.
6. EMD by Brockhoff et al. [4] employs a sliding window approach with local multi-activity feature extraction and the Earth Mover's Distance.
7. Lcdd by Lin et al. [10] uses both static and adaptive sliding windows, incorporates directly-follows relations, and ensures local completeness.

Evaluation measures. We report on results obtained using established evaluation measures for change point detection [2]. Specifically, for each event log, we compare the sequences of detected $P^d = \langle p_1^d, \ldots, p_n^d \rangle$ and gold-standard $P^g = \langle p_1^g, \ldots, p_m^g \rangle$ change points ($n, m \geq 0$), where each change point is represented by the ordinal number of the first trace that started after the change.

To identify which gold-standard change points have been successfully detected, we use the linear program proposed by Adams et al. [2] to establish a pairwise mapping between the points in P^d and P^g. This program finds an optimal mapping M, assigning as many points to each other as possible, while minimizing the distance between corresponding change points. Note that no point in P^d is assigned to multiple points in P^g or vice versa. Furthermore, M will only include pairs $p_i^d \sim p_j^g$ that are within an acceptable distance from each other, which we refer to as the allowed *latency*. We define latency as a percentage of the total traces in Σ_L, i.e., it must hold that $|p_i^d - p_j^g| \leq |\Sigma_L| * latency$. We report on results obtained using latency levels of 1%, 2.5%, and 5%.

Due to the consideration of latency, each correspondence in M is regarded as a true positive. From this, we derive *precision* (Prc.) as $|M|/|P^d|$, i.e., the fraction of detected change points that are correct according to the gold standard, *recall* (Rec.) as $|M|/|P^g|$, i.e., the fraction of correctly detected gold-standard change points, and the F1-score as the harmonic mean of precision and recall.

Configurations. We fine-tuned the RetinaNet model used by our approach with the CDLG training and validation sets and the method and parameters described in Section 3.2. Given such a fine-tuned model, the only parameter to set for inference is the number of windows N to be used. For this, we use the default number of windows $N = 200$ for the CDLG test set, whereas we reduce it to 100 windows for the CDRIFT dataset. In this way, we account for the lower number of traces in the latter dataset, ensuring that each set captures a comparable amount of process behavior across both datasets.

When reporting on the performance of the baseline techniques, we use the parameter settings that we found to achieve the highest F1-score. To find these settings for the CDLG data set, we applied the experimental framework by Adams et al. [2], which assesses different parameter configurations, on the CDLG validation set. For the CDRIFT dataset, we ran experiments using all configurations that are tested in the Adams et al. [2] framework and report on the results obtained using the best parameter settings. The exact parameter settings used for the different techniques per dataset are detailed in our repository.

Results. In the following, we present the results obtained for change point detection for the two datasets, also focusing on different latency and noise levels.[8]

Overall results. Table 2 summarizes the change point detection results.

For the CDLG test set, our `CV4CDD` approach consistently outperforms the baselines, demonstrating F1-scores ranging from 0.76 at 1% latency to 0.80 at 5% latency. It already reaches its peak performance at just 2.5% latency, surpassing the best baseline, EMD, by 0.26. In terms of recall, our approach outperforms the baseline scores by 0.21 and 0.17 at 1% and 2.5% latencies, respectively. However, at a 5% latency, the LCDD, EMD, and RINV techniques achieve comparable recall scores, each exceeding 0.60. Despite this, they exhibit lower precision,

[8] Given the non-determinism involved in training deep learning models, we repeated the training and inference procedure of our approach five times. These repetitions yielded standard deviations of just 0.0087 in recall, 0.0029 in precision, and 0.0067 in F1-score (for CDLG test). We report on the results of the first run in the remainder.

Table 2: Overall change point detection results.

Dataset	Technique	Latency 1%			Latency 2.5%			Latency 5%		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
CDLG (test)	Bose/J	0.09	0.02	0.03	0.20	0.05	0.07	0.36	0.08	0.13
	Adwin/J	0.57	0.29	0.38	0.76	0.39	0.51	0.87	0.44	0.59
	ProGraphs	0.23	0.17	0.19	0.58	0.41	0.48	0.80	0.57	0.66
	ProDrift	0.80	0.26	0.39	**0.96**	0.31	0.47	**0.98**	0.32	0.48
	Rinv	0.39	0.40	0.40	0.51	0.52	0.52	0.61	0.61	0.61
	EMD	0.50	0.44	0.47	0.57	0.51	0.54	0.71	0.63	0.67
	Lcdd	0.29	0.38	0.33	0.38	0.50	0.44	0.52	0.68	0.59
	CV4CDD	**0.90**	**0.65**	**0.76**	0.95	**0.69**	**0.80**	0.96	**0.69**	**0.80**
CDRIFT	Bose/J	0.08	0.07	0.07	0.60	0.52	0.56	0.75	0.66	0.70
	Adwin/J	0.15	0.11	0.13	0.40	0.29	0.34	0.71	0.51	0.59
	ProGraphs	0.21	0.18	0.19	0.48	0.41	0.45	0.78	0.67	0.72
	ProDrift	**0.91**	0.32	**0.48**	**1.00**	0.35	0.52	**1.00**	0.35	0.52
	Rinv	0.01	0.00	0.00	0.23	0.18	0.20	0.47	0.36	0.41
	EMD	0.05	0.03	0.04	0.88	0.59	0.71	0.97	0.66	0.79
	Lcdd	0.00	0.01	0.01	0.02	0.04	0.02	0.24	0.64	0.35
	CV4CDD	0.44	**0.38**	0.41	0.96	**0.84**	**0.90**	**1.00**	**0.87**	**0.93**

Note: support for CDLG (test): 8547 change points, CDRIFT: 156 change points.

resulting in significantly lower F1-scores. Finally, in terms of precision, ProDrift achieves scores of over 0.96 for the 2.5% and 5% latency levels, which is slightly higher than the precision of our approach (by 0.01 and 0.02, respectively). However, its recall values are notably low, particularly in the presence of noise (detailed below), leading to F1-scores between 0.39 and 0.48, compared to the 0.80 achieved by our approach.

For the CDRIFT dataset, we obtain overall similar results. Our CV4CDD approach outperforms the baselines for 2.5% and 5% latencies, achieving F1-scores of 0.90 and 0.93, respectively. These higher values can be attributed to the fact that the CDRIFT dataset contains only sudden drifts, which are relatively easier to detect for our approach. Only at 1% latency does ProDrift achieve a slightly higher F1-score of 0.48 compared to 0.41 for our approach. The reason for the lower performance of our approach is rather technical. It is mainly attributed to the annotation of sudden drifts using bounding boxes of ± 5 windows spanning around the position of an actual sudden drift in an image. In scenarios with 100 windows and a latency of just 1%, inaccuracies arise during the transformation from the coordinates of the bounding box to the corresponding window index and subsequently to the first trace within the window, leading to low precision and recall. This is supported by the correctly positioned bounding boxes in the respective images, along with the observation that accuracy sharply increases to its peak values at the next latency of 2.5%.

Noise impact. To evaluate the robustness of our approach, we report the results for the event logs with different noise levels in the CDLG test set (using 5% latency), summarized in Table 3.

Table 3: Noise impact on change point detection.

Technique	W/o noise			With 30% noise			With 60% noise		
	Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
Bose/J	0.33	0.09	0.14	0.42	0.08	0.14	0.40	0.07	0.12
Adwin/J	0.86	0.44	0.58	0.87	0.41	0.56	0.89	0.47	0.62
ProGraphSs	0.75	0.56	0.64	0.82	0.58	0.68	0.89	0.57	0.70
ProDrift	**0.98**	0.62	0.76	**1.00**	0.01	0.01	0.00	0.00	0.00
Rinv	0.79	0.82	0.80	0.40	0.38	0.39	0.43	0.43	0.43
EMD	0.71	0.64	0.67	0.72	0.63	0.67	0.71	0.63	0.67
Lcdd	0.78	**0.83**	0.80	0.33	0.61	0.43	0.35	0.44	0.39
`CV4CDD`	0.96	0.70	**0.81**	0.96	**0.70**	**0.81**	**0.95**	**0.66**	**0.78**

Results for the CDLG test set, 5% latency, with support: 8547 change points.

Our `CV4CDD` approach maintains consistent performance regardless of noise, achieving the highest F1-scores from 0.81 for logs without noise to 0.78 for logs with 60% noisy traces. In noise-free conditions, three baselines (ProDrift, Lccd, and Rinv) come close to our results. The Lccd technique shows an outstanding recall of 0.83, while ProDrift maintains its lead in precision, also seen for the noise-free CDRIFT dataset. However, all three of these baselines experience a notable decline in accuracy when noise is introduced, particularly ProDrift. Conversely, baselines with lower accuracy on the noise-free logs, for instance, EMD and ProGraphs, demonstrate less vulnerability to noise. This reveals that the baselines are subject to a trade-off between performance in noise-free conditions and robustness to noise, which does not apply to our approach.

4.3 Experiment 2: Concept Drift Detection

This section discusses the experiment we conducted to evaluate the performance of our approach to detect sudden and gradual drifts in comparison to a state-of-the-art baseline technique. This experiment includes a combined evaluation of both the change point detection and change characterization tasks. In the following, we discuss the evaluation setup and obtained results.

Evaluation Setup. First, we provide information regarding the baseline, configurations, and evaluation measures used to characterize performance accuracy.

Baseline. We compare our approach against the ProDrift technique proposed by Maaradji et al. [15]. We selected this technique because it stands out as the only existing technique that focuses on the automated detection of both sudden and gradual drifts in event logs, without requiring a manual indication of the

drift type to be searched (e.g., in contrast to BOSE/J [3]). For our experiment, we use the stand-alone Java implementation PRODRIFT 2.5[9].

Configurations. We use the same configuration for our `CV4CDD` approach as for Experiment 1, i.e., with 200 windows for the CLDG test set.

For the PRODRIFT 2.5 baseline, we follow their tool manual and executed the tool with a run-based drift detection mechanism with activated parameters for parallel search of sudden and gradual drifts. We conducted tests with different window sizes (50, 100, 150, 200, 250, or 300 traces) along with adaptive windowing, which enables the tool to dynamically adjust the window size to balance detection accuracy and drift detection latency. The best results were obtained using a window size of 150, for which we report the results in the remainder.

Evaluation measures. We report on precision, recall, and F1-score by comparing a collection of detected drifts D^d to the gold-standard drifts D^g.

Given detected $D^d(s) \subseteq D^d$ and gold standard sudden drifts $D^g(s) \subseteq D^g$, the comparison is the same as for change point detection (Section 4.2), which we use to track true positives (tp), false positives (fp), and false negatives (fn).

For gradual drifts, we consider that their detection involves the correct recognition of two change points. Naturally, given detected $D^d(g) \subseteq D^d$ and gold standard gradual drifts $D^g(g) \subseteq D^g$, a true positive (tp) occurs if there is a detected gradual drift $d_k^d \in D^d(g)$ of which both the start and end change points correspond to those of a gold-standard gradual drift $d_l^g \in D^g(g)$ (given a certain latency level). However, if only the start or end point of d_l^g is detected correctly, we still count it as 0.5 of a true positive (as well as 0.5 of a false positive).

Given these scores, we compute *precision* (Prc.) as $tp/(tp+fp)$, *recall* (Rec.) as $tp/(fn+tp)$, and the F1-score per drift type, as well for the overall detection (using weights to account for their different support values).

Results. Table 4 presents the sudden and gradual drift detection results obtained for our approach and the baseline on the CDLG test set.

Our `CV4CDD` approach achieves an overall weighted F1-score ranging from 0.77 to 0.81 across various latency levels, which aligns with the change point detection accuracy results in Experiment 1. Specifically, for sudden drifts, precision consistently surpasses 0.98 across all latencies, while recall remains around 0.76. For gradual drifts, we observe the lowest precision of 0.90 at a 1% latency, although recall ranges between 0.55 and 0.61. In contrast, the baseline's weighted F1-scores are notably lower, reaching approximately 0.26 for 5% latency. This is mainly because the baseline fails to detect gradual drifts, scoring close to 0 at all latency levels.

We identified several factors that may contribute to the low performance of the baseline. First, the CDLG test set includes more varied drift scenarios than were used in the original evaluation of ProDrift [15], thus providing a more representative assessment of its performance. Specifically, our evaluation includes logs of different sizes, numbers and types of drifts per log, change severities, and noise levels. Among others, since ProDrift fails to detect change points in the

[9] Available at http://apromore.org/platform/tools

Table 4: Overall sudden and gradual drift detection results obtained on the CDLG test set.

Technique	Drift	Latency 1%			Latency 2.5%			Latency 5%		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
CV4CDD	Sudden	0.98	0.75	0.85	0.99	0.76	0.86	0.99	0.76	0.86
(Our approach)	Gradual	0.90	0.55	0.68	0.99	0.60	0.75	1.00	0.61	0.76
	Overall	**0.94**	**0.65**	**0.77**	**0.99**	**0.67**	**0.81**	**1.00**	**0.69**	**0.81**
ProDrift	Sudden	0.27	0.35	0.30	0.41	0.54	0.47	0.46	0.60	0.52
(Baseline)	Gradual	0.01	0.00	0.00	0.01	0.00	0.00	0.04	0.00	0.00
	Overall	0.14	0.18	0.15	0.21	0.27	0.23	0.25	0.30	0.26

Support for sudden drifts: 2835, gradual drifts: 2856.

presence of noise (see Experiment 1), it thus also fails to detect drifts (both sudden and gradual) for these logs. Second, the evaluation in the original paper reported the accuracy obtained using an optimal oscillation filter parameter, which mitigates the impact of infrequent events or data gaps on the detection accuracy. However, the public ProDrift tool does not allow us to control this parameter, potentially affecting accuracy. Finally, we consider stricter detection latencies, given the importance of timely drift detection.

When examining event logs without drifts, our CV4CDD approach demonstrates a reasonable precision of 0.76 and a recall of 1.00 across all latency levels. The baseline achieves similar recall, but a precision of about 0.59. This indicates that the baseline is more prone to mistakenly detect drifts in event logs, especially those with noise, that do not actually have any.
Overall, our CV4CDD approach showcases a considerable improvement in the detection and characterization of process change points, demonstrating superior performance in latency and noise robustness compared to existing techniques.

5 Related Work

Since establishing the problem and importance of concept drift detection in process mining more than a decade ago [24], various techniques have been proposed to address this problem, as highlighted in recent literature reviews [7,21]. In this section, we discuss such existing techniques, specifically those that focus on the same tasks that we tackle with our approach.

Table 5 provides an overview of such techniques, including their scope and automation level. As shown, all selected techniques can detect change points. They use a wide range of ways to tackle this task, including statistical testing, various kinds of feature representations, windowing strategies, and clustering (see Section 4.2 for more details on the most important techniques). Many of the techniques achieve good results, as demonstrated in a recent evaluation framework [2]. However, our evaluation experiments demonstrate that our proposed

Table 5: Classification of different concept drift detection techniques.

Technique	Change point detection	Change characterization
Various works [4, 9, 10, 14, 17, 22, 26]	✷	
Yeshchenko at el. [25]	✷	(✷)
Bose et al. [3]	✷	(✷)
Martushev et al. [16]	✷	(✷)
Maaradji et al. [15]	✷	✷
Our approach (CV4CDD)	✷	✷

Legend: "✷" - automated, "(✷)" - semi-automated.

CV4CDD approach outperforms them. Furthermore, the majority of existing techniques do not go beyond the detection of change points, meaning that they are only able to recognize *when* a process's behavior significantly changed, but they do not provide insights into *how* that change manifested itself.

As shown in Table 5, a few techniques provide an exception to this, since they aim to both detect and characterize changes (though often only semi-automatically). Yeshchenko et al. [25] introduced Visual Drift Detection (VDD), which automatically identifies change points based on grouping similar declarative behavioral constraints, yet its capacity is limited due to the manual interpretation required for identifying gradual drifts. Bose et al. [3] proposed a technique for automated detection of sudden and gradual drifts using statistical testing of feature vectors, but it necessitates users to pre-specify the drift type and manually select relevant features. Martushev et al. [16] enhanced this by introducing adaptive windowing, though users still need to specify drift types beforehand. Maaradji et al. [15] introduced ProDrift, a technique that addresses previous limitations by offering automated detection of both sudden and gradual drifts. However, its statistically-based design struggles to detect gradual drifts, especially for noisy event logs, as evidenced in our evaluation. By contrast, our approach does not face such limitations, greatly outperform ProDrift in terms of detection accuracy and latency.

6 Conclusion

In this paper, we proposed CV4CDD, a novel, computer vision approach for concept drift detection in event logs. Our approach is based on an object detection model (RetinaNet) that was fine-tuned using a large collection of event logs with known concept drifts. In the conducted experiments, we demonstrated that CV4CDD considerably outperforms available baselines for change point detection and characterization on several datasets, including well-established datasets commonly used to evaluate concept drift detection techniques. Here it is worth noting that CV4CDD stands out not only as the first approach using techniques from computer vision for concept drift detection in process mining, but as the first approach using supervised machine learning in general.

In future work, we aim to enhance our approach and expand its scope and capability. We plan to conduct sensitivity analyses on various design decisions, such as window size, behavioral representations, and similarity measures, to better understand their impact on the achieved accuracy. Our ultimate objective here is to automate the selection of appropriate configurations for these design decisions to achieve the optimal visual representation of an event log for detecting concept drift. Furthermore, we intend to broaden the scope of our work by aiming to identify and characterize more complex drift types in event logs, such as incremental and recurring drifts that encompass multiple interconnected process changes. Additionally, we plan to extend concept drift detection to other process perspectives, such as time, resources, and data.

Acknowledgment

We acknowledge the work of Jonathan Kößler for conducting the initial testing of the paper's idea in his master's thesis.

References

1. van der Aalst, W.: Process Mining: Data Science in Action. Springer (2016)
2. Adams, J.N., Pitsch, C., Brockhoff, T., van der Aalst, W.: An experimental evaluation of process concept drift detection. Proc. VLDB Endow. **16**(8) (2023)
3. Bose, R.J.C., Van Der Aalst, W., Žliobaitė, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Netw. Learn. Syst. **25**(1), 154–171 (2013)
4. Brockhoff, T., Uysal, M.S., van der Aalst, W.: Time-aware concept drift detection using the earth mover's distance. In: Proc. ICPM. pp. 33–40. IEEE (2020)
5. Ceravolo, P., Tavares, G.M., Junior, S.B., Damiani, E.: Evaluation goals for online process mining: A concept drift perspective. IEEE Trans. Serv. Comput. **15**(4), 2473–2489 (2022)
6. van Der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. CSRD **23**, 99–113 (2009)
7. Elkhawaga, G., Abuelkheir, M., Barakat, S.I., Riad, A.M., Reichert, M.: CONDA-PM: a systematic review and framework for concept drift analysis in process mining. Algorithms **13**(7), 161 (2020)
8. Grimm, J., Kraus, A., van der Aa, H.: CDLG: A tool for the generation of event logs with concept drifts. In: BPM Demos. vol. 3216, pp. 92–96. CEUR-WS (2022)
9. Hompes, B., Buijs, J.C., van der Aalst, W., Dixit, P.M., Buurman, J.: Detecting changes in process behavior using comparative case clustering. In: CEUR-WS proceedings of SIMPDA. pp. 54–75. Springer (2015)
10. Lin, L., Wen, L., Lin, L., Pei, J., Yang, H.: LCDD: detecting business process drifts based on local completeness. IEEE Trans. Serv. Comput. **15**(4), 2086–2099 (2020)
11. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proc. IEEE Comput. Soc. Conf. Comput. Vis. pp. 2980–2988 (2017)
12. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Proc. ECCV. pp. 740–755. Springer (2014)

13. Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M.: Deep learning for generic object detection: A survey. IJCV **128**, 261–318 (2020)
14. Lu, X., Fahland, D., van den Biggelaar, F.J., van der Aalst, W.: Detecting deviating behaviors without models. In: BPM workshop. pp. 126–139. Springer (2016)
15. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. IEEE Trans. Knowl. Data Eng. **29**(10), 2140–2154 (2017)
16. Martjushev, J., Bose, R.J.C., Van Der Aalst, W.: Change point detection and dealing with gradual and multi-order dynamics in process mining. In: Proc. BIR. pp. 161–178. Springer (2015)
17. Nguyen, H., Dumas, M., La Rosa, M., ter Hofstede, A.H.: Multi-perspective comparison of business process variants based on event logs. In: Proc. ER. pp. 449–459. Springer (2018)
18. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M., van Dongen, B.F.V.: Detecting drift from event streams of unpredictable business processes. In: Proc. ER. pp. 330–346. Springer International Publishing, Cham (2016)
19. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D., Modugno, G.: Orange: outcome-oriented predictive process monitoring based on image encoding and cnns. IEEE Access **8**, 184073–184086 (2020)
20. Pfeiffer, P., Lahann, J., Fettke, P.: Multivariate business process representation learning utilizing gramian angular fields and convolutional neural networks. In: Proc. BPM. pp. 327–344. Springer (2021)
21. Sato, D.M.V., De Freitas, S.C., Barddal, J.P., Scalabrin, E.E.: A survey on concept drift in process mining. ACM Computing Surveys **54**(9), 1–38 (2021)
22. Seeliger, A., Nolle, T., Mühlhäuser, M.: Detecting concept drift in processes using graph metrics on process graphs. In: Proc. S-BPM. vol. 9, pp. 1–10. ACM (2017)
23. Smirnov, S., Weidlich, M., Mendling, J.: Business process model abstraction based on synthesis from well-structured behavioral profiles. Int. J. Coop. Inf. Syst. **21**(01), 55–83 (2012)
24. Van Der Aalst, W., Adriansyah, A., De Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., Van Den Brand, P., Brandtjen, R., Buijs, J., et al.: Process mining manifesto. In: BPM conference. pp. 169–194. Springer (2011)
25. Yeshchenko, A., Di Ciccio, C., Mendling, J., Polyvyanyy, A.: Visual drift detection for event sequence data of business processes. IEEE Trans. Vis. Comput. Graph. **28**(8), 3050–3068 (2021)
26. Zheng, C., Wen, L., Wang, J.: Detecting process concept drifts from event logs. In: OTM Conferences. pp. 524–542. Springer (2017)