Applied Intelligence (2023) 53:27323–27343
https://doi.org/10.1007/s10489-023-04812-0

AdaDeepStream: streaming adaptation to concept evolution in deep neural networks

Lorraine Chambers[1] · Mohamed Medhat Gaber[1] · Hossein Ghomeshi[2]

Accepted: 18 June 2023 / Published online: 7 September 2023

Abstract

Typically, Deep Neural Networks (DNNs) are not responsive to changing data. Novel classes will be incorrectly labelled as a class on which the network was previously trained to recognise. Ideally, a DNN would be able to detect changing data and adapt rapidly with minimal true-labelled samples and without catastrophically forgetting previous classes. In the Online Class Incremental (OCI) field, research focuses on remembering all previously known classes. However, real-world systems are dynamic, and it is not essential to recall all classes forever. The Concept Evolution field studies the emergence of novel classes within a data stream. This paper aims to bring together these fields by analysing OCI Convolutional Neural Network (CNN) adaptation systems in a concept evolution setting by applying novel classes in patterns. Our system, termed *AdaDeepStream*, offers a dynamic concept evolution detection and CNN adaptation system using minimal true-labelled samples. We apply activations from within the CNN to fast streaming machine learning techniques. We compare two activation reduction techniques. We conduct a comprehensive experimental study and compare our novel adaptation method with four other state-of-the-art CNN adaptation methods. Our entire system is also compared to two other novel class detection and CNN adaptation methods. The results of the experiments are analysed based on accuracy, speed of inference and speed of adaptation. On accuracy, *AdaDeepStream* outperforms the next best adaptation method by 27% and the next best combined novel class detection/CNN adaptation method by 24%. On speed, *AdaDeepStream* is among the fastest to process instances and adapt.

Keywords Deep neural netowrks · Concept evolution · Streaming machine learning · Novel class detection · Deep neural network adaptation

1 Introduction

DNNs are widely used and have achieved state-of-the-art performance in static data classification tasks [1, 2]. However, data evolves in real-world scenarios and standard DNNs are not responsive to changing data. DNNs only recognize classes they are trained on. Therefore, novel classes are attributed to known labels from the training data. This will result in incorrectly classified instances. Non-responsiveness to changing data could be dangerous in safety-critical applications such as autonomous vehicles [3] and medical sensor analysis [4]. For instance, deep neural networks have been used widely to develop autonomous vehicles and various safety incidents have been reported in the media such as Google's self driving car hitting a bus [5] and a Tesla driver's fatal crash [6]. Detection and adaptation of novel classes in these circumstances is essential. In this paper we focus on the key challenge of detecting novel classes emerging in streaming images and CNN adaptation to this.

In dynamically changing and non-stationary environments, data drift can manifest when out of distribution instances occur. When novel classes appear in a datastream in accumulated instances, this data drift is called concept evolution [7]. When there is a distribution change within the

Mohamed Medhat Gaber and Hossein Ghomeshi are both contributed equally to this work.

✉ Lorraine Chambers
lorraine.chambers@mail.bcu.ac.uk

Mohamed Medhat Gaber
mohamed.gaber@bcu.ac.uk

Hossein Ghomeshi
hosseinghomeshi@shipamax.com

[1] School of Computing and Digital Technology, Birmingham City University, 15 Bartholomew Row, Birmingham B5 5JU, Birmingham, United Kingdom

[2] Shipamax Ltd., 241 Southwark Bridge Rd, London SE1 6FP, United Kingdom

existing classes only and no new classes arise, this data drift is called concept drift [8]. This paper focuses on concept evolution only.

Detecting and adapting to concept evolution in CNNs can be problematic. In the streaming environment there is no prior knowledge of the novel classes to assist in the initial training process [9]. Image data is high dimensional which makes it harder to detect concept evolution than for lower dimensional datasets [10]. In CNN adaptation, high data dimensionality contributes to the adaptation latency being large [11]. DNN adaptation can cause catastrophic forgetting (where originally known classes are forgotten in the presence of new classes) [12]. DNNs require balanced classes for training which are not available in streaming scenarios. DNNs require a larger amount of data for training as compared to other types of machine learning models. To achieve online CNN adaptation, solutions often require prior selection of specialised DNN architectures, loss functions or knowledge distillation [13–16]. This means that novel class detection and adaptation cannot be retrospectively applied without retraining the CNN. This is particularly an issue in CNNs that take a long time to train. The requirement of online adaptation may not always be realised at the time of original implementation of the CNN system. If such a system is not in place, metrics would need to be manually monitored, data would need to be collected, labelled and another model statically trained before being updated on the system, which may not be achievable when the system is online. Therefore, easily implementable solutions to upgrade existing trained CNNs are required. At the very least, an image recognition system should be able to detect and adapt to novel classes in a datastream. For an image recognition system involving a DNN, the two fields of Concept Evolution and OCI are involved. OCI is a subset of the Online Continuous Learning (OCL) field concerning novel classes only. OCI typically trains a newly initialised DNN in an incremental manner, applying all instances of one class at a time [12]. Research in this field focuses on accumulating and preserving knowledge without forgetting any previous data [12], True-labelled samples of the classes are often used, which results in artificially high performance. In a real-world scenario, even partially labelling a data stream using humans can be expensive [17] and impractical due to the need for domain experts and manual labelling. This is in contrast to the Concept Evolution field where focus is upon the changing data, taking into account only some previous data and using minimal true-labelled samples. The Concept Evolution and OCI fields require bringing together as they offer complimentary views. We apply concept evolution patterns of abrupt, gradual, incremental and reoccurring as shown in Fig. 1.

Existing approaches for CNN adaptation to concept evolution are limited as the focus has been on lower dimensional data. To handle high dimensional data, the data is transformed into a different representation to improve the separation of the classes [10, 18–20]. Clustering is commonly used in novel class detection [10, 20–25]. It is an implicit method of drift detection. The drift is monitored over a number of instances before drift is declared. This is in contrast to explicit drift detection where the change is detected and immediately reported. Implicit methods of drift detection result in a delay of concept evolution detection. In data stream classification, there are approaches for novel class detection in images where DNNs are used in the detection process [20, 26–28]. However, not where a CNN is adapted to the novel classes. This means that CNN adaptation has not been studied in depth in the concept evolution field. On the contrary, the OCI field has a large number of CNN adaptation techniques [14–16, 29, 30]. However, they have not been applied to the more dynamic concept evolution setting. Only one paper [21] started to address this by unifying OCL with concept drift and concept evolution adaptation [21]; however, data drift patterns were not applied. Our own previous work uses the activations from within a DNN with streaming clustering, but had limitations with memory [31] and are placed in static setting [32]. Thus, a new solution is required that addresses concept evolution detection and CNN adaptation to images with the following attributes: (1) Transformation of the image data into a different representation. (2) Explicit concept evolution detection. (3) A solution that can be applied to existing CNNs with no need to re-train the CNN. (4) Analysis with respect to concept evolution patterns.

Our system (*AdaDeepStream*) aims to provide a wrapper whereby pre-trained standard CNNs can be enabled to explicitly detect and adapt to concept evolution in images. The concept evolution detection and CNN adaptation is facilitated by the use of activations from within the deep neural network, applied to streaming machine learning models. Our system detects change in an unsupervised manner by comparing the predictions of the CNN with predictions from a streaming classifier using the internal activations of the CNN. Only instances that are detected at the beginning of the change are true-labelled to minimise human interaction. The aim is to adapt a CNN within seconds.

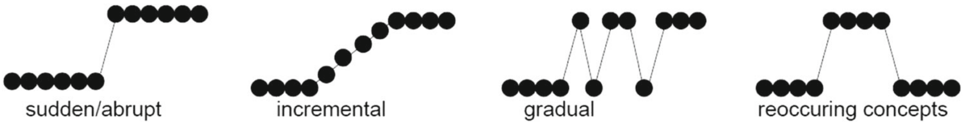


Fig. 1 Patterns of concept evolution applied over time. Figure adapted from

This research focuses on image classification via the widely used VGG16 Convolutional Neural Network [33]. Datasets CIFAR-10, CIFAR-100 [34] and Fashion-MNIST [35] are used. Concept evolution patterns are produced by withholding classes from the dataset during training, then applying unseen instances of known and withheld classes during testing. We analyse unseen instances by utilising activations from the hidden layers within the deep neural network. We reduce the activations using two methods (1) an extension of our Jensen-Shannon Divergence (JS-Divergence) as used in [32], altered to also calculate JS-Divergence between each layer and the final layer. This is referred to as JSDL for the remainder of this paper. (2) a modified Content-Based Image Retrieval (CBIR) method [36], referred to as DS-CBIR for the remainder of this paper. We apply the activations to a Hoeffding Adaptive Tree [37] streaming classifier. The difference between the Hoeffding adaptive classifier and the CNN is used to detect concept evolution via DDM (Drift Detection Method) [38]. Once concept evolution is detected, the CNN is adapted via our method, termed DSAdapt. We substitute our DNN adaptation method with four leading methods from the OCI field: iCARL (Incremental Classifier and Representation Learning) [14], LwF (Learning without Forgetting) [15], ER (Experience Replay) [29, 30] and MIR (Maximally Interfered Retrieval) [16] augmented with RV (Review Trick) [39]. We perform an extensive empirical study, comparing these and our DSAdapt method based on accuracy, speed of inference, and speed of adaptation. Our entire system, *AdaDeepStream*, is an offline training and online inference method. Its effectiveness is shown in comparison to RSB (Reactive Subspace Buffer) [21] and CPE (CNN based Prototype Ensemble) [10] also through accuracy, speed of inference and speed of adaptation.

The CBIR technique [36] is not ours but is modified to remove the threshold and further reduce the activations by splitting them into equal sections and averaging each section. To the best of our knowledge, applying CBIR to concept evolution detection is unique. JSDL uses the JS-divergence statistical difference measure in a novel way between layers of a deep neural network. To the best of our knowledge, apart from our own previous work in [31, 32]; using the reduced activations for concept evolution detection is unique. Our CNN adaptation (DS-Adapt) is novel. Reduced activations provide a buffer of training and true-labelled instances, assisting in addressing class imbalance and catastrophic forgetting. To the best of our knowledge, using activations with streaming machine learning models for CNN adaptation is novel. A summary of the contributions of this paper is as follows:

1. Heuristics for activation reduction of deep neural networks via DS-CBIR and JSDL to apply to concept evolution detection.
2. Concept evolution detection using neural network activations and streaming machine learning models.
3. CNN adaptation involving neural network activations and streaming machine learning models.
4. Analysis of OCI CNN adaptation techniques in a concept evolution setting.

This paper is organised as follows: In Section 2 we discuss the related work. In Section 3, we present a description of the system that includes formalisation and implementation details of the *AdaDeepStream* components and methodology. In the experimental study in Section 4 we specify the experimental setup. In Section 5, we evaluate *AdaDeepStream* with four other CNN adaptation methods and two combined novel class detection and CNN adaptation solution. Section 6 summarises our findings with a conclusion and suggestions for future work.

2 Related work

In this section, the work related to concept evolution, drift detection and DNN adaptation in the concept evolution and OCI fields are discussed.

2.1 Concept evolution

Concept evolution is usually detected via distance measures based on instances' locations from the decision boundaries. OLINDDA [40] and MINAS [41] are based on clustering. ECSMiner [42] and Echo-D [43] are based on ensembles of classifiers with hyper-spheres. The aforementioned methods have to wait for a number of novel classes defined by a threshold before a novel class is declared. Distance-based methods are difficult to get to work on high dimensional data such as image data and suffer the curse of dimensionality [44]. A method to counteract this is to project the data into a different representation. To detect novel classes, methods such as temperature scaling [18], perturbations or input pre-processing [19] and learning a latent feature embedding space [10, 20] are employed to separate the classes and make the novel class detection more feasible. In this paper, instead of using the image data directly, the activation data from the hidden layers of the deep neural network is extracted and used to train a streaming classifier, offering a reduced alternative representation of the image data. The predictions of the CNN and the streaming classifier are presented to a drift detector and monitored for change in order to detect novel classes. Therefore, the following section contains related work concerned with drift detection.

2.2 Drift detection

Drift detection algorithms can be categorised into (1) sequential, (2) adaptive windowing and (3) statistical [45–47]. Sequential based methods analyse whether the drift has occurred and sequentially and predict the drift based on the accuracy evaluation. Examples are Cumulative Sum (CUSUM) [48] and, Geometric Moving Average (GMA) [49]. CUSUM calculates the difference of observed values from the mean and raises an alarm when it is significantly different. GMA uses a forgetting factor to weight the latest data and a threshold is used to tune the false alarm rate. However, these algorithms require tuning of the parameter values, resulting in a trade-off between false alarms and detecting true drifts. Windowing methods use fixed or dynamic windows that summarize information; that information is then used to make a comparison between the previously summarized windows and the current window to detect drift. Examples of adaptive windowing methods are Adaptive Windowing (ADWIN) [50] and Kosmolgorov Simirnov Windows (KSWIN) [51]. ADWIN uses sliding windows of variable size and if two windows are found that have distinctly different averages, then the data distribution is deemed to have changed. KSWIN is based on the Kosmolgorov statistical test and has no assumption of the underlying data distribution. Another adaptive windowing methods is HDDM [52] which is based on Hoeffding bounds and monitors the data distribution of different time windows using probability inequalities instead of the probability distribution function. It compares the moving averages to detect the drifts and uses a forgetting scheme to find the weight of moving averages in the data stream. However, it needs to explore with different weighting schemes for application to real-world problems [47]. Statistical methods are based on parameters like mean and standard deviation to predict drift. Examples are DDM [38] and EDDM [53]. DDM is based on the PAC learning model [38]. It assumes the binomial distribution and uses the standard deviation to detect drift and works well on abrupt drift. EDDM improves upon DDM by increasing the detection of gradual drift whilst maintaining a good abrupt drift detection rate by using a distance error rate instead of the classifiers error rate. The statistical methods are generally faster than the sequential or adaptive windowing methods, and although DDM is one of the older algorithms, it is one of the most accurate and fastest [54].

2.3 Concept evolution and DNN adaptation

Adaptation of DNNs in the concept evolution field is relatively new, as the focus has tended to be on traditional machine learning models, because DNNs typically require a large amount of training data and their adaptation latency is large [11]. The main challenges faced in adapting DNNs are (1) which parts of the network should be static and which parts plastic [55], (2) sample shortage, and (3) the time to train the DNN [56]. Existing concept evolution and DNN adaptation methods can be divided into two categories (1) model parameter updating and (2) model structure updating [11]. The update of the model parameters preserves the structure of the network and alters parameters such as the weights of the network to adapt to concept evolution. All of the parameters in the network are affected, thus it can be slow to adapt [57]. Partial parameter updating is targeted to specific areas of the DNN, leading to faster adaptation and less catastrophic forgetting. The parts of the network that are updated can vary. In [58] the layer is found where a drift first occurs and subsequent layers only are updated for drift adaptation. Model structure updating involves adjusting the network width or depth, often leading to complex models which increases training time. An example of this is [13] in which small networks are added to the existing structure to accommodate the new classes. There are methods that use deep learning methods to detect concept evolution [20, 26, 27], but few that detect concept evolution and adapt a pre-existing CNN. In [21] an experience replay method is used, where experience replay means that previously seen instances are stored for later use. A reactive subspace buffer tracks novel class occurrences such that the buffers remember valid information and forget outdated information and split buffers based on novel class detection.

2.4 Online class incremental

In OCI, the aim is to accumulate and preserve knowledge without forgetting any previous data [12]. This is known as catastrophic forgetting, where the network cannot perform well on previously seen data after updating with new data[59]. Methods in the OCI field are based on: (1) Regularization, (2) Memory and (3) Parameter Isolation. Most methods are based on Memory. Methods that excel in performance in the OCI setting are iCARL, LwF, MIR and ER [12]. LwF is regularization based and all others are memory-based. LwF uses knowledge distillation [18] to preserve knowledge from past tasks and uses a teacher/student model. However, the teacher/student model means it has reliance on relatedness between the old and the new data and may not perform well if the distributions of the old and new data are different [15]. iCARL (Incremental Classifier and Representation Learning) creates a training set by mixing all of the samples in the memory buffer and the current unseen samples, adjusts the loss function to address class imbalance between old and new classes and uses a nearest-class-mean (NCM) classifier instead of a Softmax classifier. iCARL has the best performance with a small memory buffer on small datasets. ER simply trains the model with the unseen data batch and memory mini-batches, has an efficient training time and has outperformed other approaches [29]. MIR (Maximally Interfered Retrieval) selects memory samples according to the loss increases given the estimated parameter update based on the incoming unseen window and updates the DNN on this sample plus the original unseen window. It performs well on large scale datasets with a large memory buffer [12]. There are also a number of "tricks" that can be applied that can enhance existing adaptation methods. These address class imbalance due to the strong bias towards the novel classes. Examples of the strongest ones are Review Trick (RV) [39], which uses an additional tuning step with a small learning rate and a memory buffer and NCM [60], which replaces the fully connected layer and the Softmax layer with a Nearest Class Mean classifier. However, inference time increases with the growth of the memory size whereas for RV, the training time increases with the growth of the memory size but it is more memory-efficient than NCM [12].

2.5 Novel class detection and adaptation in high dimensional data

In summary, to the best of our knowledge, there is only one system that detects novel classes and directly adapts the classifying CNN (RSB [21]). Other systems have DNNs involved in the detection of novel classes (CPE [10], CSIM [20]). There are a larger number of other methods that adapt to high dimensional novel classes that do not involve a DNN (EMC [22], SAND [23], SENNE [24], KNNENS [25], ECSMiner [42], SEND [61], SENCForest [62], SACCOS [63], SENC-Mas [64]). Table 1 shows a summary of approaches for novel class detection and adaptation in high dimensional data. From this, it can be seen that most solutions for adapting to novel classes in high dimensional data use clustering. However, this is implicit drift detection. Therefore, the drift detection requires a number of instances before the novel class is declared, resulting in a delayed adaptation response. Our system differs from these as it does not use clustering in the drift detection, but a tree and an explicit change detection method. Our method adapts the image classifying DNN. There is a lack of other methods focusing upon this. Therefore, we investigated OCI DNN adaptation methods. These are summarised in Table 2. Some of these methods require re-training of the DNN on the initial training data as they use specialised architectures, loss functions or knowledge distillation. Similarly to the most successful adaptation methods, we use a memory-based approach, but use activations to assist in the adaptation. The concept for this paper is that the *AdaDeepStream* sys-

Table 1 Summary of approaches for novel class detection and adaptation in streaming high dimensional data

Approach	Method
Clustering. Adapting DNN.	RSB: Centroid clustering. Reactive subspace buffer tracks drift. DNN adaptation is Memory-based. Stores diverse samples.
Clustering. Uses DNN in detection.	CPE: Changes representation of images into clustering prototypes to improve cohesion and separation. Uses a CNN in the novel class detection.
	CSIM: CNN used to form feature embedding to learn high level features of images and transformed into a space with improved cohesion and separation using metric learning, then novel class detection is performed.
Clustering. No DNN.	EMC: Evolving micro-clusters with local density novel class detector.
Clustering Ensemble. No DNN.	SAND: Clustering K-NN ensemble and detects outliers having strong cohesion among themselves.
	SENNE: Nearest Neighbour ensembles. Calculates geometric distance between classes.
	Echo-D: Ensemble of clustering classifiers.
Clustering Ensemble. No DNN.	KNNENS: k-nearest neighbor-based hypersphere ensemble.
Clustering Tree/Forest. No DNN.	ECSMiner: Decision Tree and Feature space partitioning.
Clustering Tree/Forest, No DNN.	SEND: Clustering random trees.
Tree/Forest. No DNN.	SENCForest: Completely random trees with an unsupervised anomaly detector.
Clustering Graph. No DNN.	SACCOS: Mutual graph clustering.
Matrix Sketching. No DNN.	SENC-MaS: Class Matrix Sketching to model the data as a low-dimension approximation.
Tree. Adapting DNN.	*AdaDeepStream* (Ours): Hoeffding tree using activations and DNN prediction applied to explicit drift detector. DNN adaptation is memory-based involving DNN activations.

Table 2 Summary of approaches for OCI DNN adaptation

Approach	Method
Memory. Re-training required	iCARL: Centroid Memory buffer with reservoir sampling. Updated Loss function and distillation loss. Binary cross-entropy between old and new class
	MIR: Estimated parameter update based on the incoming batch using stochastic gradient descent, using samples from the memory buffer that have the largest estimates as loss increases and mixes them with the incoming mini-batch
Memory. No re-training	ER: Replay method that applied reservoir sampling and trains the model with the incoming data and memory data using cross entropy loss
	AdaDeepStream (Ours): Trains model with incoming data windows where novel classes have been detected. Samples data from DNN activation training data and true-labelled data.
Regularization. Re-training required	LwF: A student model is created and the original model becomes the teacher. Uses knowledge distillation.

tem can accept a standard pre-trained CNN and adapt it to novel classes. For drift detection we adopt a statistical method to ensure fast detection of concept evolution. For the CNN adaptation, we adopt a partial model parameter update methodology for the reasons of complexity affecting training time, making only the classification layers of the CNN plastic so that it will adapt quickly, and use a memory-based methodology for providing samples to the adaptation. Section 3 provides an in-depth description of our *AdaDeepStream* system.

3 AdaDeepStream system

This section details our *AdaDeepStream* system components and their interactions via descriptions and algorithms. We present our two activation reduction methods (JSDL and DS-CBIR), our drift detection method, our CNN adaptation method (DS-Adapt), and how the application of the four substituted CNN adaptation methods is achieved.

Firstly, some fundamental concepts are given. For the definition of a datastream, let $D = \{(x_i, y_i)\}_1^\infty$ where (x_i, y_i) is an instance that has arrived at timestep t. $x_i \in R^d$ where R^d is a d-dimensional feature vector. y_i is the true class label of the instance. Concept evolution are new classes that emerge during inference. They are not present in the initial training data. For data stream classification with novel class detection: Let D_{init} be a training set of m examples: $D_{\text{init}} = \{(x_i, y_i)\}_1^m$ where the class labels are: $y_i \in Y = \{1, 2, \ldots, L\}$ and Y are the known class labels. An initial model is built with: $f : X \rightarrow Y$. The model f, is then used to predict the class labels of incoming instances if it belongs to a known class, or flag up a change. Each change detection is considered a possible new class in that window. The model is then adapted to include the novel class. In the following algorithm descriptions, index notation is used. For instance, x_{ih} is the ith instance in hidden layer h. Table 3 lists the commonly used symbols. For brevity in the descriptions, the Hoeffding Adaptive Tree streaming activation classifier is referred to as HAT, JS-Divergence is referred to as JSD and the SAMKNN streaming clusterer is referred to as SAM.

Figure 2 shows the proposed *AdaDeepStream* system. The prerequisites for the system are: (1) a trained CNN on which the detection and adaptation is being applied, and (2) the data instances on which the deep neural network is trained. *AdaDeepStream* is an offline-online system in that it is trained offline and processes unseen instances online. To train *AdaDeepStream*, the training instances are presented to the CNN. The activations of each training instance are extracted, then the activation data proceeds through 3 stages: (1) activation reduction, (2) drift detection setup and (3) adaptation setup. Activation reduction is achieved via JSDL (Section 3.1) or via DS-CBIR (Section 3.2). The HAT and SAM are trained with reduced activation data: $A_{\text{init}} = \{(a_i, y_i)\}_1^m$. Where A_{init} is a training set of m samples of activation data, where a_i is an instance of reduced activation data. The class buffer is initialised with image and activation data: $E_{\text{init}} = \{(x_i, a_i, y_i)\}_1^m$. Where E_{init} is a training set of m samples of image and activation data. At inference time, a window of image data, D_w is applied. The reduced activations are extracted via JSDL or DS-CBIR to give A_w. Each instance in A_w is classified by the HAT to give a window of HAT predictions, S_w. The CNN predictions, P_w are compared with S_w. Instances where the HAT and CNN predictions match are assigned as 0. Instances where the HAT and CNN prediction do not match are assigned as 1. These are provided to the DDM drift detector. If the drift detector identifies changes, the window data, (A_w, D_w) is passed to the adaptation phase. In Fig. 2, units involving the adaptation phase are shown in orange. The window that change was

Table 3 Summary of symbols

Symbol	Description	Symbol	Description
A_{init}	Activation training data	J_m	Maximum activation values of each channel
a_i	Reduced activation data for an instance	J_w	Hidden layer activations for window
A_s	Sampled activation data from class buffer	K_w	SAMKNN predicted values for window
A_w	Reduced activations for window	L	Total number of classes
b	Convolutional block	l	Known class
B	Set of change detection windows	l	Known class
b_p	Pooling layer of convolutional block	m	Number of samples
B_p	Previous window buffer	M	Maximum number of activation values in a layer
C	CNN	P	CNN predicted results for datastream
C_a	Adapted CNN	p_i	CNN predicted class label for an instance
D	Datastream	P_w	CNN class predictions for window
D_{init}	Image training data	q_i	Drift detector output
d_i	Drift detector input	Q_w	Set of drift detector outputs for window
D_s	Sampled image data from class buffer	s_i	HAT predicted class lable for an instance
D_w	Data window	S_w	HAT class predictions for window
E	Class buffer	U	Set of stored windows
E_{init}	Class buffer of initial training data	w	Window number
G	Number of windows in U	W	Total number of windows in buffer
h	Hidden layer number	w	Window number
H	Total number of hidden layers	W	Total number of windows in buffer
j	Hidden layer	x_i	An instance of image data
J_{br}	Section averaged convolutional block activations	Y	All known classes
J_{hr}	Section averaged final hidden layer activations	y_i	True class label for an instance
J_b	Set of activation values for convolutional blocks	Y_n	Novel classes
J_c	Activation values of convolutional layer	Y_s	Sampled class labels from class buffer
J_H	Final hidden layer activations	Y_w	True class labels for window
J_i	Activations for an instance		

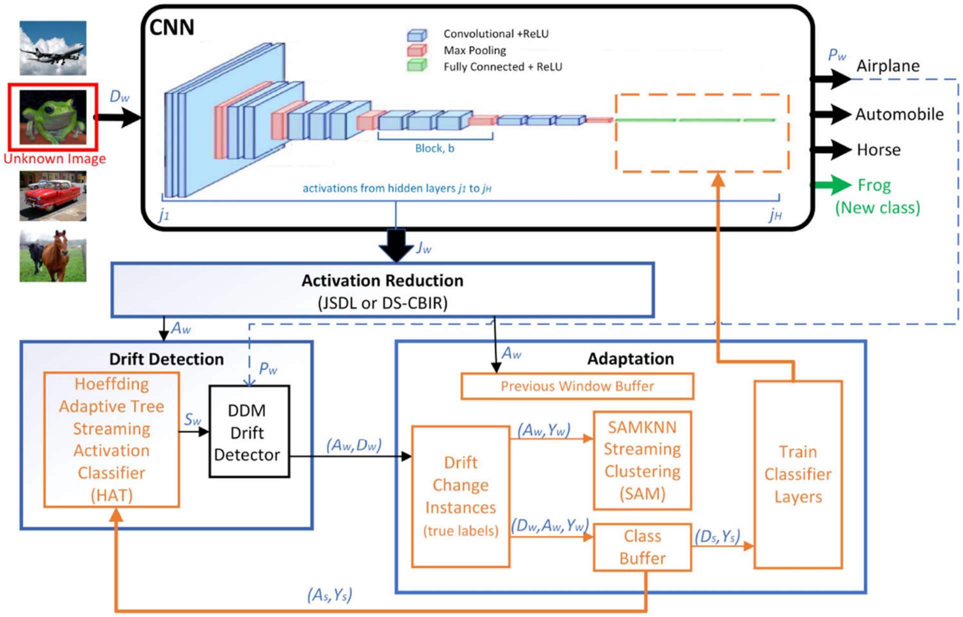


Fig. 2 *AdaDeepStream* System overview. An unknown image of a frog is presented to the CNN, the activations are extracted, drift is detected and the CNN is adapted to recognise the new class. CNN image adapted from [65]

detected in is true labelled. The SAM and the class buffer are updated with the true labelled window data. Windows prior to the drift detection are collected from the previous window buffer. Samples are taken from the class buffer for each class detected in the previous window buffer to assist in mitigating catastrophic forgetting. The sampled image data (D_s, Ys) is applied to the adaptation of the CNN. Only the CNN classification layers are trained as shown by the dashed orange box in the CNN. If the CNN is adapting when new windows, Dw are arriving, the instance labels stored in the previous window use the prediction from SAM instead of from the CNN as SAM has already been updated with the novel classes. While the CNN is adapted, the HAT is adapted with the activation instances from the class buffer (A_s, Y_s) to recognise i.e. the Frog class. From this point on the frog is a recognised class and does not trigger the drift detector.

3.1 Activation reduction JS-diverge last (JSDL)

We calculate the JS-Divergence between each pair of consecutive hidden layers and between each layer and the final hidden layer. (1), (2), and (3) show how JS-Divergence is calculated in order to achieve a dynamic representation of the activations of a deep neural network. The symbols for the following equations are standard mathematical notation and are defined in the following text, not in Table 3.

$$H = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) \tag{1}$$

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i)) \tag{2}$$

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2} + \frac{1}{2} D_{KL}(q||\frac{p+q}{2}) \tag{3}$$

JS-Divergence is a smooth symmetric measure of the similarity between two probability distributions based on the Kullback-Leibler divergence (KL-Divergence). KL-Divergence D_{KL} is based on entropy (a quantification of how much information is in data). H is the entropy and if log_2 is used, entropy is the minimum number of bits it would take to encode the information. For instance, if p is the probability distribution of the activations in hidden layer 1, we get an entropy measure of H bits from hidden layer 1. The KL-Divergence combines two probability distributions and calculates the difference of the log values for each. For instance, if q is our distribution of the activations from hidden layer 2, KL-Divergence calculates how much information is lost when p is compared with q. However, KL divergence cannot be used to measure the distance between two distributions as it is not symmetric. The JS-Divergence D_{JS} calculates a normalized score that is symmetrical. The JS-Divergence has been selected as the statistical difference measure as it, or the KL-Divergence has been used in conjunction with DNN activations [66, 67]. Other statistical measures of Kolmogorov-Smirnov and cosine similarity were experimented with. However, JS-Divergence provided superior results. To compare layers using JS-Divergence, the number of activations utilised from each neighbouring layer needs to be the same. The largest layer size in the CNN is selected and if the layer sizes are smaller, they are padded with zero's, then the JS-Divergence is calculated between the neighbouring layers. This yields one value between each activation layer per data instance. The algorithm for our JSDL implementation is presented in Algorithm 1. In the overall *AdaDeepStream* system (Algorithm 3), JSDL would be called in Line 2 via the reduceActivations method.

For previously unseen instances arriving at the deep neural network, the activations for one layer at a time is extracted via **blue**getActivations (Line 6). The JS-Divergence is calculated for two neighbouring layers; therefore, the first layer is skipped as shown in Line 7. The layers are flattened into a 1-D array via **flatten** in (Line 8) and padded to the size of the largest layer in the network in **pad** (Line 9) and the JS-Divergence is calculated between the neighbouring layers in **jsdiverge** (Line 10). This JS-Divergence calculation process is repeated for the current layer and the final layer in **jsdiverge** (Line 11). This is repeated for all layers. The set of reduced activations (A_w) for the window is returned.

3.2 CBIR activation reduction - DS-CBIR

CBIR [36] is intended for content-based image retrieval. It creates descriptors for images using deep neural networks. It is based on obtaining neural codes from fully connected layers activations. CBIR takes this one stage further by using the information contained in convolutional layers. However, the number of neurons in the convolutional part is large and most of them do not contribute significantly to the final classification. Therefore the most significant neuron activations only are extracted in order to provide extra information about the image such as background textures or colour distribution that is present in the convolutional layers [36]. We have modified CBIR to use within *AdaDeepStream* to extract the most useful activations from the network such that we can utilise it in our streaming classifier. The algorithm for our CBIR implementation is presented in Algorithm 2. We have called this DS-CBIR to distinguish it from the original. This is invoked in the overall *AdaDeepStream* system (Algorithm 3) Line 2, via the **reduceActivations** method.

For previously unseen instances arriving at the deep neural network, the activations for one block at a time is extracted, where a block in a CNN consists of convolutional layers, then a pooling layer. For each channel in the block, the maximum

Algorithm 1 JSDL

Input: One window of image data, D_w
Input: CNN, C expressed in hidden layers: $h \in C = \{1, 2, \dots, H\}$
Output: A_w: One window of reduced activations

```
1: let $M$ be the max number of activations in all layers of $C$
2: let $H$ be the number of the final hidden layer
3: for $x_i \in D_w$ do
4:     let layer number, $h = 0$
5:     for $h \in C$ do
6:         $J_{ih} \leftarrow$ getActivations($h$) ▷ get activations for current instance and hidden layer
7:         if $h > 1$ then
8:             $J_{ih} \leftarrow$ flatten($J_{ih}$)    ▷ convert activations into 1-D array
9:             $J_{ih} \leftarrow$ pad($J_{ih}$, $J_{in-1}$, $M$)    ▷ pad adjacent activation layers to the same size
10:            $J_{ih} \leftarrow$jsdiverge($J_{ih}$, $J_{ih-1}$)    ▷ calculate JSD for adjacent layers
11:            $A_w \leftarrow$jsdiverge($J_{ih}$, $J_{iH}$)    ▷ calculate JSD for current and last layer
12:        end if
13:        h = h + 1
14:    end for
15: end for
16: Return $A_w$
```

Algorithm 2 DS-CBIR

Input: One window of image data, D_x
Input: CNN, C expressed in convolutional blocks: $b \in C = \{1, 2, \dots, N\}$
Output: A_w: One window of reduced activations

```
1: for $x_i \in D_x$ do
2:     for $b \in C$ do
3:         let $b_p$ be the pooling layer of block $b$
4:         for each channel, $c$ of $b_p$ do
5:             $J_m \leftarrow$ max($b_p$) ▷ get max channel value in pooling layer, no threshold
6:         end for
7:         $J_c \leftarrow$ getConvLayer($b$)    ▷ get conv layer 1 values, no threshold
8:         Add $J_m$ and $J_c$ to set $J_b$
9:     end for
10:    $J_H \leftarrow$ getActivations(H)    ▷ get final hidden layer activations
11:    $J_{br} \leftarrow$ sectionAvg($J_b$, 16)    ▷ get the average value for 16 sections
12:    $J_{hr} \leftarrow$ sectionAvg($J_H$, 32)    ▷ get the average value for 32 sections
13:    $A_w \leftarrow J_{br} + J_{hr}$
14: end for
```

activations for the pooling layer is extracted via **max** (Line 5). Corresponding values are obtained from the first convolutional layer in the block via the **getConvLayer** method (Line 7). This is repeated for each block in the network. Due to the constraint on the number of input features the streaming classifier can accept, we have modified CBIR. We only extract the values from the first convolutional layer in each block. The original CBIR paper [36] used a threshold to save on computing time, reasoning that as ReLU (Rectified Linear Unit) activation functions were used, then processing under an activation threshold of 0.5 was not advantageous. As our system is designed to be flexible for different types of CNNs, we removed this threshold which, on our system, did not incur a significant increase in computing time but improved the clustering of the activations. Lines 2 to 10 is the original CBIR algorithm [36]. Additionally, the combined output of the max pooling layer activations and the convolutional layer activations from each block are reduced further via the **sectionAvg** method (Line 11) where set C_i is split into 16 equal parts and the average for each part is calculated. The activations for the last hidden layer J_H are extracted and these are also reduced into 32 values in the same way (Line 12). These reduced sets are combined (Line 13) and returned as the set of reduced activations for the window. This is required so that the number of values presented to the streaming classifiers are within the range of the number of input features accepted by the streaming classifier.

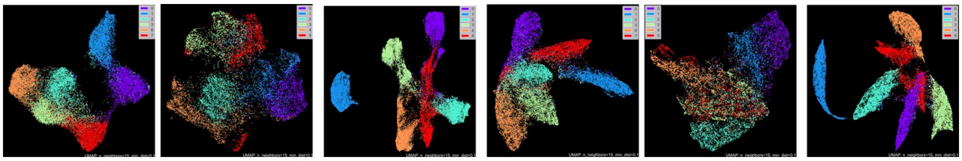
To achieve DS-CBIR, the convolutional blocks need to be identified for the CNN. This is done automatically by *AdaDeepStream*, but may need manually adjusting, whereas JSDL simply uses all of the networks layers. Figure 3 shows UMAP [68] representations of the reduced activations. From this we can see that Fashion-MNIST activation data potentially has superior clustering and separation of the classes than the CIFAR-10 and CIFAR-100 data.

3.3 Drift detection method

DDM (Drift Detection Method) [38] is used for drift detection. The drift detection method in *AdaDeepStream* can be substituted for any other drift detection method. An empirical study was conducted with other drift detectors. As only drift detection and not more information such as severity or region of the drift is required, error-based drift detection methods only were analysed. Statistical control-based and data distribution window-based drift detectors were selected. The statistical based drift detectors of DDM [38] and EDDM [53] outperformed the data distribution window-based drift detectors of ADWIN [50], KSWIN [51], HDDMA and HDDMW methods [52]. DDM was significantly faster than EDDM, therefore it was selected.

After the activations have been reduced, they are supplied to a Hoeffding Adaptive Tree streaming classifier which is established and well used [37]. The Hoeffding Adaptive Tree is trained on the reduced activations obtained from the training data. Algorithm 3 shows the overall *AdaDeepStream* process at inference time. For previously unseen instances arriving in the datastream, the prediction is obtained from the CNN, **cnnPredict** (Line 3). The reduced activations are extracted and a prediction from these is obtained from the Hoeffding Adaptive Tree Classifier, **hatPredict** (Line 4). The predictions are then compared. If they match, 0 is provided to the drift detector, otherwise 1 is provided (Lines 5 to 15). The drift detector returns a 'W' if a change warning is detected, and returns a 'C' if a change is detected. This is stored for later use (Line 14). If a warning or a change is detected in the window, then the true labels for that window are obtained, **getTrueValues** (Line 17). In reality, the images in this window would be displayed to the user for them to label them

Fig. 3 UMAP representations of reduced activation training data for six classes in the CIFAR-10, CIFAR-100 and Fashion-MNIST datasets



(a) CIFAR-10 JSDL (b) CIFAR-100 JSDL (c) Fashion-MNIST JSDL (d) CIFAR-10 DS-CBIR (e) CIFAR-100 DS-CBIR (f) Fashion-MNIST DS-CBIR

Algorithm 3 *AdaDeepStream*

```
Input: Windows of image data $D$
Input: Pre-trained CNN, $C$ on $L$ classes
Input: Pre-trained Hoeffding Adaptive Tree on activations of $L$ classes
Output: $P$: A set of CNN predicted classes for datastream

1: for $D_w \in D$ do
2:     $A_w \leftarrow$ reduceActivations($D_w$)    ▷ reduce activation via JSDL or DS-CBIR
3:     $P_w \leftarrow$ cnnPredict($D_w$)    ▷ get CNN predictions
4:     $S_w \leftarrow$ hatPredict($A_w$)    ▷ get HAT predictions
5:     for $x_i \in D_w$ do
6:         let $a_i$ be the reduced activations for $x_i$
7:         let $p_i$ be the CNN predicted class for $x_i$
8:         let $s_i$ be the HAT predicted class for $x_i$
9:         if $p_i = s_i$ then    ▷ check if CNN and HAT predictions match
10:            $d_i = 0$    ▷ 0 if match
11:        else
12:            $d_i = 1$    ▷ otherwise 1
13:        end if
14:        $Q_w \leftarrow$ driftDetector($d_i$)    ▷ detect drift change
15:    end for
16:    if 'W' or 'C' is in $Q_w$ then    ▷ drift detector returns 'W' for warning or 'C' for change
17:        $Y_w \leftarrow$ getTrueValues($D_w$)    ▷ get true values for the drift detection window
18:        $Y \leftarrow Y + Y_n$    ▷ add new labels to known classes
19:        $C \leftarrow$ adaptDnn($D_w, A_w, Y_w$)    ▷ adapt CNN via DS-Adapt or another method
20:    else
21:        Let $U$ be the window buffer and $G$ be the number of windows in $U$
22:        if CNN adaptation in progress then
23:            $K_w \leftarrow$ samPredict($A_w$)    ▷ use SAM predictions if CNN adapting
24:            $U \leftarrow (D_w, A_w, K_w)$
25:        else
26:            $U \leftarrow (D_w, A_w, P_w)$    ▷ use CNN predictions
27:        end if
28:        if $G > 2$ then    ▷ only store 2 windows
29:            $U \leftarrow U - U_0$    ▷ remove oldest window
30:        end if
31:    end if
32:    $P \leftarrow P_w$
33: end for
```

with the true values. As only the windows where the change was detected are displayed to the user, there is less labelling than if all the true values are used. The next step is adaptation (Line 19). Image data D_w, reduced activation data A_w and their true labels Y_w are provided to the **adaptDNN** method. This method can be any CNN adaptation method. Our CNN adaptation algorithm, DSAdapt, is described in Algorithm 4. We substituted this adaptation with four other state-of-the-art methods. The results are shown in Section 5. If there is no drift detected, the previous G windows are stored in U (Lines 28 and 29). If there is CNN adaptation occurring in the background, the window is re-predicted via the SAM clusterer, **samPredict** and stored (Lines 23 and 24). As the SAM clusterer is the first to be updated with true labels, this improves the accuracy of the stored windows whilst the CNN adaptation is awaited. If there is no CNN adaptation occurring, the window is stored with the CNN predictions (Line 26). SAMKNN is a popular Self Adjusting Memory (SAM) model for the k Nearest Neighbor (kNN) [69, 70]. It operates on a window of instances which is set to 1000 instances; therefore, the memory usage will not increase over time.

3.4 CNN adaptation

CNN adaptation is based on transfer learning with clustering, a class buffer and some memory of previous instances. Our CNN adaptation method DSAdapt is described in Algorithm 4. When a warning of a change or a change is detected, the window of data within which it is detected is provided to the CNN adaptation method. The current window and the previous 2 windows are added to a buffer (Line 2). These are the change detection windows. If the buffer exceeds 5 groups of change detection windows, the first group in the buffer is removed (Line 4). This provides some 'memory' for the CNN adaptation. True-labelled instances are supplied to the SAMKNN clusterer [71] (Line 6). The buffer instances are re-predicted via the SAMKNN Streaming Classifier (Line 7) to give K_w. True-labelled instances are also supplied to a class buffer that stores training data instances of the original image data and the equivalent activation data (Line 9). Each time a window is added to the class buffer, the oldest window is removed (Line 10), ensuring that the size of the class buffer does not increase. Each class in K_w and the true values for the change detection window Y_w is randomly sampled from the class buffer. One hundred instances (or if there is not enough, the maximum number of instances that are available) are randomly selected from the class buffer (Lines 11 to 13). The change detection windows, B are added to the sampled class buffer data (Line 14). The CNN is adapted for 3 epochs (Line 15). The current

Algorithm 4 *AdaDeepStream* Adaptation Method - DSAdapt

```
Input: One window of image data: $D_w$
Input: One window of reduced activation data $A_w$
Input: One window of true labels: $Y_w$
Input: CNN to be adapted, $C$
Input: Buffer of two previous windows, $U$
Output: Adapted CNN, $C_a$

1: let $W$ be the total number of change detection groups in buffer, $B$
2: $B \leftarrow U + (D_w, A_w, Y_w)$    ▷ save the current change detection and previous two windows
3: if $W > 5$ then    ▷ if there's more than 5 groups of change detection buffers
4:     $B \leftarrow B - B_0$    ▷ remove oldest group
5: end if
6: samPartialFit($A_w, Y_w$)    ▷ add true labelled data to SAM
7: $K_w \leftarrow$ samPredict($B$)
8: Let $E$ be the class buffer
9: $E \leftarrow E + (D_w, A_w, Y_w)$    ▷ Add true labelled data to class buffer
10: $E \leftarrow E - (D_0, A_0, Y_0)$    ▷ remove oldest instance
11: for class $l$ in $(K_w + Y_w)$ do    ▷ get 100 samples per class from buffer
12:     $(D_s, A_s, Y_s) \leftarrow$ getClassBufferSample($l$, 100)
13: end for
14: $(D_s, A_s, Y_s) \leftarrow (D_s, A_s, Y_s) + B$    ▷ add change detection windows to samples
15: copy C and train with $(D_s, Y_s)$ for 3 epochs    ▷ adapt CNN
16: hatPartialFit($A_s, Y_s$)    ▷ adapt HAT
17: replace $C$ with adapted CNN, $C_a$
18: repeat steps 1 to 17 on background thread
```

CNN is replaced with the adapted CNN (Line 17). Once the adaptation has been triggered, it continues adapting (Lines 1 to 17) on a background thread, using the current instance window and previous windows predicted via the SAMKNN Streaming Clusterer (Line 18), replacing the true values Y_w with the predicted values from the clusterer K_w. In summary, to achieve a balance between avoiding catastrophic forgetting and remembering recent data only, windows pertaining to the last five change detections are stored. Only the classes found in these change detection windows are sampled from the class buffer. Therefore, not all previous classes that have ever arrived are used in the adaptation, but only more recent ones. As a true-labelled instance is added to the class buffer, the oldest instance is removed, ensuring the size of the class buffer does not increase over time. Catastrophic forgetting is partially mitigated by remembering the recent classes only. This is intentional as in the concept evolution field it is not the aim to remember all classes.

4 Experimental study

In this section, we present our experimental setup. We use datasets CIFAR-10, CIFAR-100 [34] and Fashion-MNIST [35]. The CIFAR-10 dataset consists of 10 different classes of 32 × 32 colour images. In total there are 50000 training images and 10000 test images. The CIFAR-100 dataset consists of 100 different classes of 32 × 32 colour images. The 100 classes are grouped into 20 super-classes. The super-classes are used in this paper and are listed in Table 4. In total there are 5000 training images and 1000 test images. There are 2500 training images per super-class, and 500 test images per super-class. The Fashion-MNIST dataset consists of 10 different classes of 28 × 28 greyscale images. In total there are 60000 training images and 10000 test images. Each of the classes is assigned a class number ranging from 0 to 9. Three combinations of eight trained classes and two novel classes have been selected from each dataset. An empirical study was conducted on the effectiveness of our drift detection mechanism on different combinations of pairs of novel classes. It was found that the drift detection was more effective when the classes differed in their categories. For CIFAR-10, the classes can be split into categories of Transport and Animals, and for Fashion-MNIST, the classes can be split into categories of Clothing and Footwear. Therefore, to give a more rounded analysis, novel class combinations were selected as (1) a pair containing a mix of categories, (2) a pair containing classes from the same category (Animals for CIFAR-10 and Clothing for Fashion-MNIST) and (3) a pair containing classes from the other category (Transport for CIFAR-10 and Footwear for Fashion-MNIST). The selected combinations are shown in Table 4. For CIFAR-100, the class combinations were randomly selected. The trained and novel class identifiers are listed with a key of the class labels.

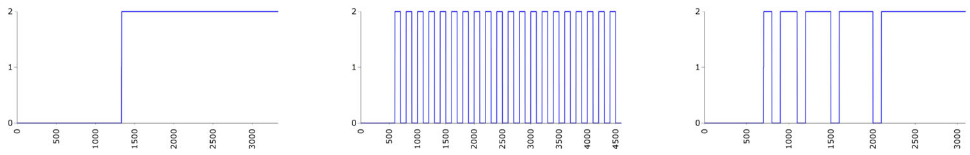
The CNN that has been applied to the proposed system is VGG16, using transfer learning from ImageNet weights and trained on three combinations of classes for each data setup. All hidden layers that have measurable outputs in Pytorch [72] are used. We selected a well-known pre-trained network and applied transfer learning as this is a common scenario in real-world applications.

For the test data, the instances are applied as in Figs. 4 and 5. The x-axis is the number of images, and the y-axis is the cumulative number of concept evolution classes that have been introduced into the datastream. For instance, in Figs. 4 and 5, when the line is at zero, this represents that only images the CNN has been trained on are in the data stream, and when the line is at 1, all concept evolution classes have been applied to the stream. Figure 5 shows the categorical patterns. There are two steps, representing that there were two concept evolution classes that have been cumulatively applied at different times in the datastream. No incorrectly classified instances are removed during testing in order to simulate real-world applications.

The CNN adaptation of *AdaDeepStream* is substituted with four different adaptation methods: (1) LwF [15] (2) iCARL [14], (3) ER [29, 30] and (4) MIR [16], augmented

Table 4 Class data combinations

Trained Classes	Novel Classes	Class Identification Key
Fashion-MNIST		
0-1-2-3-5-6-8-9	4-7	0 = T-Shirt/Top
2-3-4-5-6-7-8-9	0-1	1 = Trousers
0-1-2-3-4-6-7-8	5-9	2 = Pullover
0-1-2-3-4-6	5-7-8-9	3 = Dress
4-5-6-7-8-9	0-1-2-3	4 = Coat
2-3-4-6-8-9	0-1-5-7	5 = Sandal
0-1-8-9	2-3-4-5-6-7	6 = Shirt
0-1-5-6	2-3-4-7-8-9	7 = Sneaker
0-1-2-7	3-4-5-6-8-9	8 = Bag
6-9	0-1-2-3-4-5-7-8	
1-8	0-2-3-4-5-6-7-9	
0-5	1-2-3-4-6-7-8-9	
CIFAR-10		
0-1-2-3-4-6-7-8	5-9	0 = Airplane
2-3-4-5-6-7-8-9	0-1	1 = Automobile
0-1-2-4-5-6-8-9	3-7	2 = Bird
0-1-2-3-6-7	4-5-8-9	3 = Cat
2-3-4-5-6-7	0-1-8-9	4 = Deer
0-1-4-5-8-9	2-3-6-7	5 = Dog
5-7-8-9	0-1-2-3-4-6	6 = Frog
0-2-6-7	1-3-4-5-8-9	7 = Horse
1-2-3-4	0-5-6-7-8-9	8 = Ship
0-7	1-2-3-4-5-6-8-9	9 = Truck
1-8	0-2-3-4-5-6-7-9	
2-3	0-1-4-5-6-7-8-9	
CIFAR-100		
1-3-4-5-6-7-10-11-13-17	8-18	0 = Aquatic Mammals
0-1-3-5-11-12-15-17-18-19	2-7	1 = Fish
1-5-7-8-9-14-15-16-17-18	0-10	2 = Flowers
4-7-8-9-11-14-15-17-18-19	2-5-6-16	3 = Food Containers
0-2-4-5-7-9-13-14-15-18	10-11-12-19	4 = Fruit and Vegetables
5-6-7-10-11-12-14-17-18-19	1-4-8-9	5 = Household Electrical Devices
1-2-4-6-7-9-11-16-18-19	0-8-10-12-13-17	6 = Household Furniture
0-1-6-7-8-9-11-12-17-18	2-4-5-15-16-19	7 = Insects
0-2-3-5-8-9-10-11-12-19	4-6-13-15-17-18	8 = Large Carnivores
0-9-10-11-12-13-14-16-17-18	2-3-4-5-6-8-15-19	9 = Large Man-Made Outdoor Things
0-2-3-4-6-8-9-13-16-19	1-10-11-12-14-15-17-18	10 = Large Natural Outdoor Scenes
2-5-10-11-13-15-16-17-18-19	1-3-6-7-8-9-12-14	11 = Large Omnivores and Herbivores
4-5-8-12-14-15-16-17-18-19	0-1-2-3-6-7-9-10-11-13	12 = Medium-Sized Mammals
1-2-3-4-5-6-7-8-12-13	0-9-10-11-14-15-16-17-18-19	13 = Non-Insect Invertibrates
0-1-7-8-9-10-11-14-18-19	2-3-4-5-6-12-13-15-16-17	14 = People
		15 = Reptiles
		16 = Small Mammals
		17 = Trees
		18 = Vehicles 1
		19 = Vehicles 2

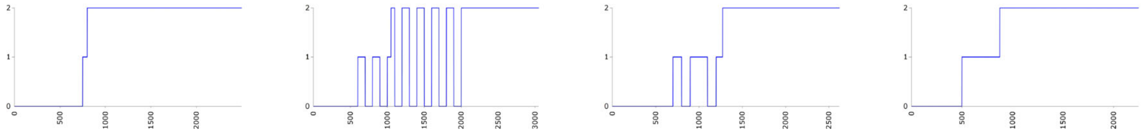


(a) Abrupt (b) Reoccurring (c) Gradual

Fig. 4 Temporal types of concept evolution patterns

with trick RV [39]. These are successful methods in the OCI setting. Some methods are adjusted to be appropriate for the single-head setting. Single-head [73] configuration is where all classes (previously known and novel classes) have a single shared output layer and do not need to know the class label [12]. The alternative is multi-head where more than one output layer is created as the model adapts and extra information is required to select the correct head. Our method is a single-head configuration and we will therefore compare with the single-head implementations. Each of these methods has been adjusted to receive only the true-labelled windows of instances when drift was detected from *AdaDeepStream* and to remove prior knowledge of the number of novel classes. The following hyper-parameters, as specified in [12] are set for all of the CNN adaptation methods: learning rate = 0.01, epochs = 3, weight decay = 0 and memory buffers = 5000 (the mid-range that was used in the survey paper [12]). The entire *AdaDeepStream* system is compared to RSB [21] and CPE [10]. Implementations of these methods are available along with our *AdaDeepStream* code at the location referenced in the Code and Data Availability Section. An overview of each of these methods follows.

1. LwF (Learning without Forgetting) is a regularization-based method and uses knowledge distillation [18] to remember past tasks. A student model is created for the new task and the original model becomes the teacher model. A variant of LwF is used LwF.MC that only has one head (where all tasks share the same output head) [14].
2. iCARL (Incremental Classifier and Representation Learning) is a memory-based method. A training set is constructed by mixing all the samples in the memory buffer and the current task samples. The loss function has a classification loss to help the model correctly predict the novel classes and a knowledge distillation loss to prompt the model to reproduce the outputs from the previous model for old classes. It uses binary cross-entropy for each class to handle the imbalance between old and new classes. Originally, an NCM classifier is used with the memory buffer to predict labels for test images [60], which means it looks for a subset of samples whose mean of latent features have the closest Euclidean distance to the mean of all samples in this class; however, this method requires all samples from all classes, and therefore cannot be applied in the online setting. Therefore, the modified version from [12] with reservoir sampling [74] is used instead of NCM [14].
3. ER (Experience Replay) is a memory-based method that applies reservoir sampling [74] and random sampling. ER trains the model with the incoming data and memory batches using the cross-entropy loss.
4. MIR (Maximally Interfered Retrieval) is a recent memory-based method that performs an estimated parameter update based on the incoming mini-batch using stochastic gradient descent. It uses the samples from the memory buffer that have the largest estimated loss increases and mixes them with the incoming mini-batch. In this paper, MIR is augmented with RV (Review Trick) [39]. as RV can provide an improvement and is more efficient in memory than NCM [12]. RV reduces class imbalance by applying a fine-tuning step with a small learning rate, using a balanced subset from the memory buffer and the training set [75]. As recommended, We use RV from [75] with a learning rate 10 times smaller than the training learning rate.
5. RSB (Reactive Subspace Buffer) is a memory-based method. It has centroid-driven memory and stores diverse samples of incrementally arriving instances. A reactive sub space buffer tracks drift occurrences in previously seen classes and adapts clusters accordingly [21]. User



(a) Abrupt (b) Reoccurring (c) Gradual (d) Incremental

Fig. 5 Categorical types of concept evolution patterns

configurable parameters are set to: max centroids = 10, max size of class buffer = 100, window size = 100. All other parameters are as recommended in the paper [21].

6. CPE (CNN based Prototype Ensemble) is cluster-based and projects the images into a learned discriminated feature representation called prototypes. This improves the intra-class compactness and expand inter-class separateness in the output feature representation to enable the robustness of novel class detection. When a pre-defined number of novel class instances are detected, the CNN re-trains [10]. Contrary to our system and RSB, the CNN is used in the detection of the novel classes, not as an overall image classifying CNN. Recommended settings from the paper are used for the training of the CNN: 1000 instances from each class of training data, number of novel classes that are accumulated before CNN re-training = 1000, learning rate = 0.001 [10]. Tuning was performed for the threshold of the prototypes and a value of 5.0 was found to produce the best results.

We experiment with two different methods of activation reduction: JSDL and DS-CBIR. The system is running on AMD Ryzen Threadripper PRO 3955WX 16-Cores 3.90 GHz, 256 GB RAM with NVIDIA RTX A6000 GPU.

5 Experimental results

In this section, we discuss our thorough experimental study, evidencing the efficacy of the proposed method. The results are discussed in terms of accuracy for each of the activation reduction methods and subsequently, timings for the time per instance and DNN adaptation time.

Experiments were conducted on three datasets (CIFAR-10, CIFAR-100 and Fashion-MNIST). Each using two different activation reduction methods (DS-CBIR and JSDL). CIFAR-100 accuracy plots are shown in Figs. 6 and 7. Plots for all other datasets are included in Appendix A. The RSB and CPE results are displayed within the DS-CBIR plots for ease of comparison, but they do not use any activation reduction method. Figures 6 and 7 show how the accuracy of the models vary with an increasing number of applied novel classes for the CIFAR-100 dataset. Generally, the accuracy of all methods tend to decrease when the number of novel classes are increased. This is more prominent in the DS-CBIR results (Fig. 6) than it is in the JSDL results (Fig. 7), where the accuracies are more erratic. This pattern is also prominent in the other dataset results in Appendix A. This indicates that DS-CBIR is more consistent at detecting the changes in drift than JSDL. In the Temporal-Reoccurring plots in Figs. 6, 7 and Appendix A, higher accuracies are reported than for the other concept evolution patterns. This can be seen for both DS-CBIR and JSDL. This could be due to there being a more diverse range of old and new classes in less windows in this pattern as compared to the other patterns. It is temporal, so the known classes are applied randomly rather than incrementally. The novel classes occur in small blocks alongside the known classes. This more diverse data is applied to the adaptation.

As DS-CBIR is more stable than JSDL, we focus on the DS-CBIR activation reduction method. Table 5 shows the average accuracy after adaptation for DS-CBIR, all datasets and all concept evolution patterns. The equivalent data for the JSDL activation reduction method is included in Appendix A, Table 8. The data combinations with a large number of trained classes and a small number of novel classes are shown in these tables. This scenario is more usually seen in real-world applications. For instance, a model would be pre-trained on known data, and see a few novel classes arriving, such as in medical image analysis where identification of changes in chest x-rays is required. A deep neural network would be trained on many known diseases, and one or two new variations resulting in

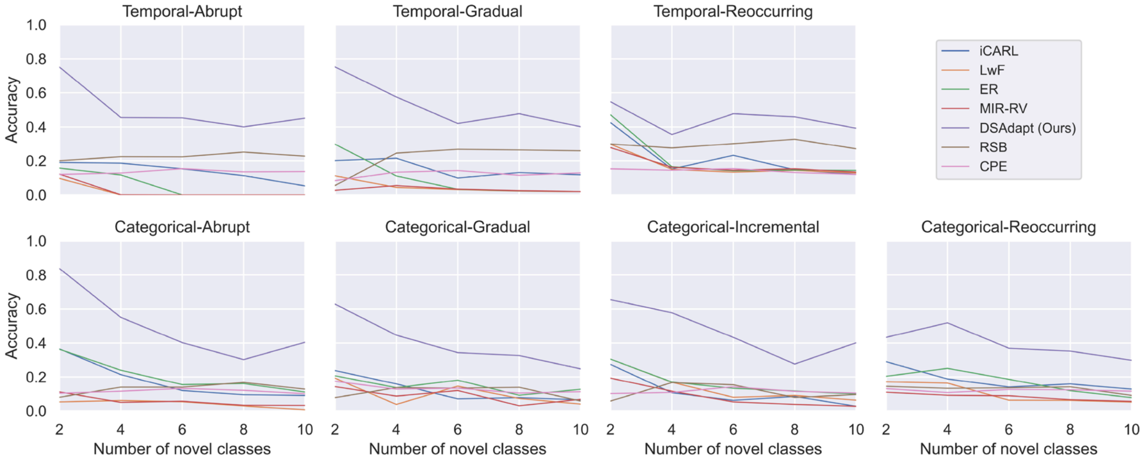


Fig. 6 Number of novel classes against accuracy for DS-CBIR VGG16 CNN, CIFAR-100 for all concept evolution patterns

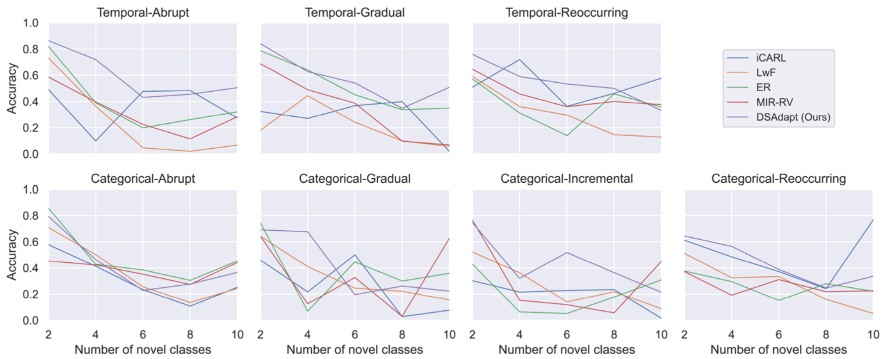


Fig. 7 Number of novel classes against Accuracy for JSDL VGG16 CNN, CIFAR-100 for all concept evolution patterns

different chest x-ray results would occur, rather than many new manifestations in a short amount of time. For DS-CBIR, CIFAR-10 and CIFAR-100, Our method (ADS) has exceeded all others, except in one scenario of Temporal-Reoccurring. The accuracies for Fashion-MNIST are generally higher for all methods. This is probably due to the superior intra-class cohesion and inter-class separation of the reduced activation data, as demonstrated in the UMAP [68] representations in Fig. 3. Our method is among the best; however, CPE excels on the Fashion-MNIST dataset. The standard deviation of the methods ranges from 0.116 to 0.684, our method has the highest. The implicit drift detection methods of RSB and CPE have standard deviations of 0.116 and 0.091 respectively, which are the lowest. Therefore, the higher standard deviation may indicate that the explicit nature of the drift detection has an effect on the consistency of our method. There is a connection between how successful the CNN adaptation is and how successful the drift detection is, as the drift detection is based on the difference in predictions between the CNN and the Hoeffding Adaptive Tree Streaming Classifier. Therefore, the less successful the adaptation is, the more unstable the drift detection will be. On average, of the different concept evolution patterns applied, our DSAdapt method outperforms the other adaptation methods. The two best-performing methods are DSAdapt (ours) and ER, which are both memory-based and use the standard cross-entropy loss. The regularization-

Table 5 Average accuracy after CNN adaptation for each concept evolution pattern for DS-CBIR activation reduction. Highest values are in bold

Reduction/ Dataset	Method	Cat. Abr.	Tem. Abr.	Cat. Gra.	Tem. Gra.	Cat. inc.	Cat. Reo.	Tem. Reo.
DS-CBIR	iCARL	0.304	0.308	0.171	0.174	0.154	0.056	0.442
CIFAR-10	LwF	0.044	0.080	0.236	0.127	0.354	0.187	0.437
8 Trained	ER	0.548	0.453	0.459	0.677	0.457	0.377	**0.694**
2 Novel	MIR-RV	0.070	0.090	0.131	0.113	0.162	0.112	0.491
classes	RSB	0.359	0.416	0.343	0.429	0.270	0.386	0.630
	CPE	0.546	0.470	0.458	0.481	0.372	0.460	0.398
	ADS (ours)	**0.888**	**0.819**	**0.543**	**0.807**	**0.732**	**0.702**	0.598
DS-CBIR	iCARL	0.365	0.190	0.238	0.201	0.274	0.290	0.425
CIFAR-100	LwF	0.053	0.097	0.193	0.112	0.305	0.172	0.298
10 Trained	ER	0.363	0.157	0.207	0.298	0.305	0.204	0.471
2 Novel	MIR-RV	0.112	0.122	0.143	0.027	0.193	0.110	0.278
classes	RSB	0.080	0.200	0.078	0.055	0.058	0.145	0.299
	CPE	0.104	0.12	0.175	0.084	0.102	0.131	0.153
	ADS (ours)	**0.838**	**0.752**	**0.629**	**0.753**	**0.655**	**0.433**	**0.547**
DS-CBIR	iCARL	0.681	0.116	0.235	0.569	0.227	0.103	0.626
MNIST-Fashion	LwF	0.180	0.049	0.106	0.115	0.183	0.102	0.461
8 Trained	ER	0.806	0.606	0.317	**0.907**	0.324	0.166	0.728
2 Novel	MIR-RV	0.18	0.167	0.198	0.205	0.001	0.038	0.547
classes	RSB	0.771	0.656	0.606	0.644	0.682	0.687	**0.816**
	CPE	0.925	**0.897**	**0.93**	0.898	**0.907**	**0.861**	0.815
	ADS (ours)	**0.964**	0.661	0.831	0.884	0.860	0.807	0.568

Table 6 Time per instance (ms) (with standard deviation in brackets) and rank. Lowest values are in bold

	DS-CBIR Time (ms)	Rank	JSDL Time (ms)	Rank
iCARL	266.5 (82.2)	6	329.5 (92.5)	6
LwF	8.1 (7.7)	3	**14.2 (8.0)**	**1**
ER	**7.1 (2.8)**	**1**	16.3 (13.5)	3
MIR-RV	7.2 (2.4)	2	16.3 (13.6)	4
RSB	19.9 (7.8)	5	-	-
CPE	545.2 (290.1)	7	-	-
ADS (Ours)	8.5 (2.1)	4	15.7 (7.0)	2

based method, LwF did not perform well in our scenarios. CPE only performs well on the Fashion-MNIST dataset. This method does not have a CNN that classifies images, but has a CNN that learns a feature representation of the images. Therefore, it is not a direct comparison and our results suggest it prefers data that has superior intra-class cohesion and inter-class separation. After *AdaDeepStream*, the next highest performing adaptation method is ER and the next best performing overall concept evolution/CNN adaptation method is CPE. On average, in our scenarios in Table 5, *AdaDeepStream* outperforms ER by 27% and CPE by 24%. The Wilcoxon Signed-Rank test is used to analyse the difference between the accuracies of methods. The difference between the accuracy of *AdaDeepStream* and that of ER over the 272 tested data patterns is statistically significant. The p-value is less than 0.00001, which is less than 0.05 significance level, suggesting the acceptance of the alternative hypothesis that true location shift is not equal to 0. The same is true for cPE and RSB. Therefore, *AdaDeepStream* with our DSAdapt adaptation method significantly outperforms the next-best substituted CNN adaptation method of ER and *AdaDeepStream* significantly outperforms the drift detection and adaptation methods of CPE and RSB.

Tables 6 and 7 summarise the timings for the inference time and the time to adapt, respectively. The time to process one instance in milliseconds is shown in Table 6. This is the time per instance, measured in batches of 100 instances. ER has the fastest time per instance with 7.1ms, with *AdaDeepStream* having the fourth fastest at 8.5ms. This is close to the fastest results given that CPE and iCARL are 545.2m and 266.5ms respectively. From the adaptation time, ours is third fastest at 7.69 seconds. The fastest is LwF at 6s, with CPE being the slowest at 420s, placing *AdaDeepStream* close to the fastest adaptation time. From Tables 6 and 7, JSDL outperforms DS-CBIR with regards to the time per instance, and adaptation time. However the drift detection is erratic.

Table 7 Adaptation time (s) (with standard deviation in brackets) and rank. Lowest values are in bold

	DS-CBIR Time (s)	Rank	JSDL Time (s)	Rank
iCARL	78.095 (45.3)	6	115.373 (53.2)	6
LwF	**6.095 (7.1)**	**1**	21.463 (23.4)	3
ER	6.774 (6.1)	2	16.976 (18.2)	2
MIR-RV	36.000 (17.9)	5	56.256 (38.1)	5
RSB	1.995 (0.7)	4	-	-
CPE	420.000 (123.0)	7	-	-
ADS (Ours)	7.690 (1.9)	3	**7.816 (1.2)**	**1**

6 Conclusion and future work

This paper places CNN adaptation methods from the OCI field in the Concept Evolution field. Our *AdaDeepStream* adaptation method overall outperforms other leading OCI adaptation methods in accuracy when placed in the concept evolution scenario with limited true-labelled data. *AdaDeepStream* also outperforms other drift detection and CNN adaptation systems (RSB and CPE) in accuracy. From the two novel activation reduction methods presented (JSDL and DS-CBIR), DS-CBIR produces more stable results than JSDL. *AdaDeepStream* performs well on all concept evolution patterns whilst other methods show an improvement on the Temporal-Reoccurring pattern, possibly due to a more diverse range of classes arriving in the drift detection windows. Compared to the other methods, *AdaDeepStream* also performs well on data with less intra-class cohesion and inter-class separation. However, it is less stable compared to the implicit drift detection comparison methods. This could indicate that the explicit drift detection and the use of CNN predictions in the drift detection method has an effect on the consistency of our method. Therefore, a good drift detection method is important. The speed of inference and adaptation of *AdaDeepStream* is comparable with the fastest adaptation methods. *AdaDeepStream* is a memory-based CNN adaptation method as is the next best performing method, ER. In this scenario of small datasets, it indicates that the simple memory-based methods achieve good results. In this paper, we applied our method to image data and a CNN. Further directions of study are: (1) apply to larger real-world datasets, (2) further explorations into the drift detection to extend it to concept drift detection, (3) providing mechanisms to assist in true labelling of samples, (4) Apply to other types of DNNs, although DS-CBIR is specific to CNNs and would need to be adjusted for the architecture of the DNN and (5) perform an ablation study to investigate what effect areas such as the activation reduction, the class buffer and the previous window buffers have upon our system.

Appendix A Accuracy Plots

Extended results plots and tables.

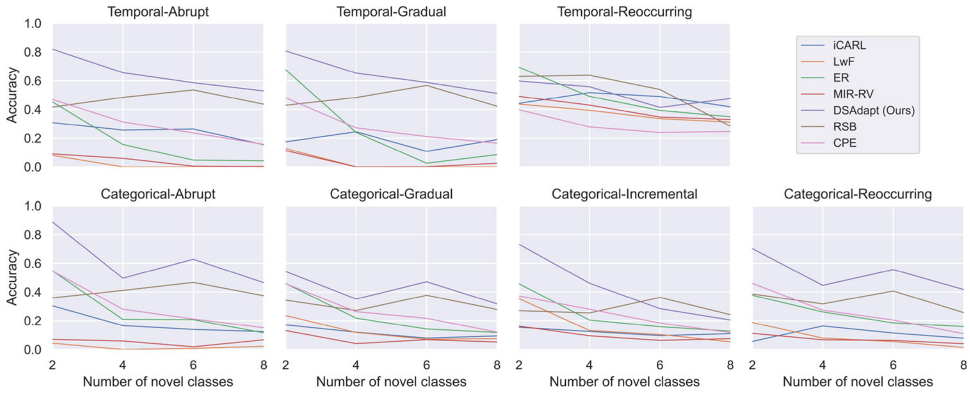


Fig. 8 Number of novel classes against accuracy for DS-CBIR, VGG16 CNN, CIFAR-10 for all concept evolution patterns

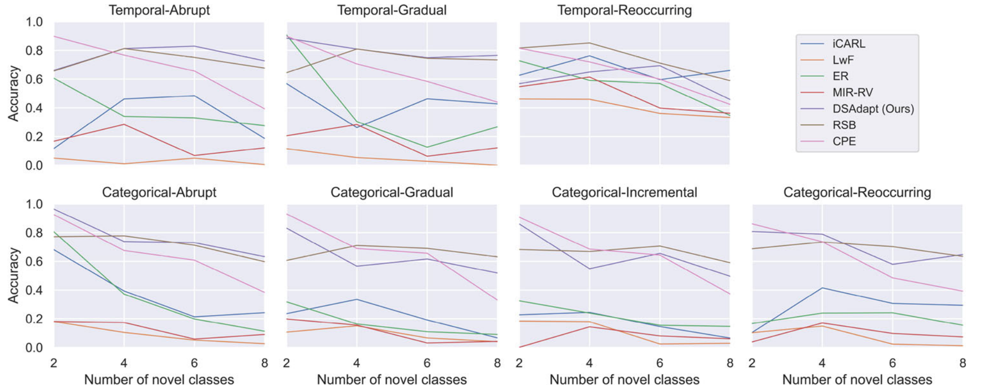


Fig. 9 Number of novel classes against accuracy for DS-CBIR, VGG16 CNN, Fashion-MNIST for all concept evolution patterns

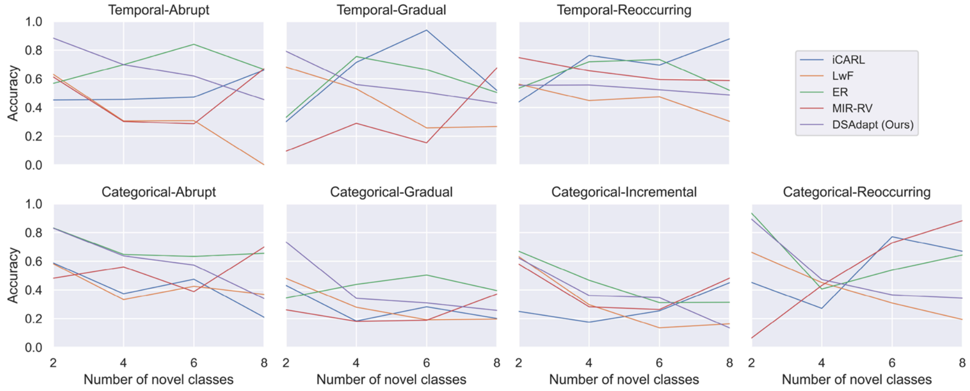


Fig. 10 Number of novel classes against accuracy for JSDL, VGG16 CNN, CIFAR-10 for all concept evolution patterns

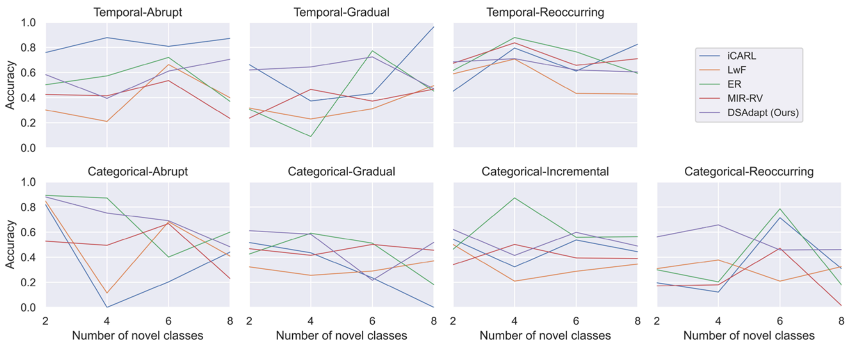


Fig. 11 Number of novel classes against accuracy for JSDL, VGG16 CNN, Fashion-MNIST for all concept evolution patterns

Table 8 Average accuracy after CNN adaptation for each concept evolution pattern for JSDL activation reduction. Highest values are in bold

Reduction/ Dataset	Method	Cat .Abr.	Tem. Abr.	Cat. Gra.	Tem. Gra.	Cat. inc.	Cat. Reo.	Tem. Reo.
JSDL	iCARL	0.588	0.452	0.431	0.301	0.25	0.452	0.438
CIFAR-10	LwF	0.582	0.631	0.48	0.682	0.632	0.662	0.563
8 Trained	ER	**0.832**	0.568	0.343	0.331	**0.669**	**0.936**	0.535
2 Novel	MIR-RV	0.481	0.613	0.261	0.095	0.580	0.064	**0.748**
classes	ADS (ours)	0.830	**0.883**	**0.734**	**0.792**	0.622	0.893	0.555
JSDL	iCARL	0.578	0.491	0.460	0.324	0.303	0.612	0.508
CIFAR-100	LwF	0.708	0.732	0.642	0.180	0.522	0.512	0.591
10 Trained	ER	**0.856**	0.821	**0.744**	0.787	0.429	0.376	0.572
2 Novel	MIR-RV	0.453	0.586	0.642	0.688	**0.765**	0.372	0.645
classes	ADS (ours)	0.793	**0.864**	0.691	**0.842**	0.746	**0.644**	**0.760**
JSDL	iCARL	0.821	**0.759**	0.516	**0.663**	0.543	0.196	0.450
MNIST-Fashion	LwF	0.848	0.302	0.322	0.317	0.501	0.308	0.589
8 Trained	ER	**0.892**	0.503	0.425	0.308	0.462	0.299	0.615
2 Novel	MIR-RV	0.527	0.425	0.467	0.236	0.339	0.171	0.674
classes	ADS (ours)	0.880	0.584	**0.611**	0.620	**0.621**	**0.562**	**0.685**

Funding The authors did not receive support from any organization for the submitted work.

Data Availability The datasets generated during and/or analysed during the current study are publicly available datasets and are downloaded upon first use via the code in the Code Availability Section

Code Availability The code is available at https://github.com/chambai/AdaDeepStream

Declarations

Conflicts of interest The authors have no conflicts of interest to declare that are relevant to the content of this article

References

1. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444. https://doi.org/10.1038/nature14539. Number: 7553 Publisher: Nature Publishing Group
2. Szegedy C, Ioffe S, Vanhoucke V, Alemi AA (2017) Inception-v4, inception-ResNet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI Press, San Francisco, California, USA, 2017), AAAI' 17, pp 4278–4284
3. Pal SK, Pramanik A, Maiti J, Mitra P (2021) Deep learning in multi-object detection and tracking: state of the art. Appl Intell 51(9):6400–6429. https://doi.org/10.1007/s10489-021-02293-7

4. Abdallah ZS, Gaber MM, Srinivasan B, Krishnaswamy S (2018) Activity recognition with evolving data streams: a review. ACM Comput Surv 51(4):71:1-71:36. https://doi.org/10.1145/3158645
5. Lee D (2016) Google self-driving car hits a bus. BBC News
6. Yadron D, Tynan D (2016) Tesla driver dies in first fatal crash while using autopilot mode. Section: technology
7. Din SU, Shao J, Kumar J, Mawuli CB, Mahmud SMH, Zhang W, Yang Q (2021) Data stream classification with novel class detection: a review, comparison and challenges. Knowl Inf Syst 63(9):2231–2276. https://doi.org/10.1007/s10115-021-01582-4
8. Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):1–37. https://doi.org/10.1145/2523813
9. Zhang Z, Li Y, Gong Y, Yang Y, Ma S, Guo X, Ercisli S (2023) Dataset and baselines for IID and OOD image classification considering data quality and evolving environments. International Journal of Interactive Multimedia and Artificial Intelligence 8(Special Issue on AI-driven Algorithms and Applications in the Dynamic and Evolving Environments)
10. Wang Z, Kong Z, Changra S, Tao H, Khan L (2019) Robust high dimensional stream classification with novel class detection. In 2019 IEEE 35th international conference on data engineering (ICDE), pp 1418–1429. https://doi.org/10.1109/ICDE.2019.00128. ISSN: 2375-026X
11. Yuan L, Li H, Xia B, Gao C, Liu M, Yuan W, You X (2022) Recent advances in concept drift adaptation methods for deep learning. In thirty-first international joint conference on artificial intelligence, vol. 6: pp 5654–5661. https://doi.org/10.24963/ijcai.2022/788. ISSN: 1045-0823
12. Mai Z, Li R, Jeong J, Quispe D, Kim H, Sanner S (2022) Online continual learning in image classification: An empirical survey. Neurocomputing 469:28–51. https://doi.org/10.1016/j.neucom.2021.10.021
13. Yoon J, Yang E, Lee J, Hwang SJ (2018) Lifelong learning with dynamically expandable networks. International Conference on Learning Representations p 11
14. Rebuffi SA, Kolesnikov A, Sperl G, Lampert CH (2017) iCaRL: incremental classifier and representation learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2001–2010
15. Li Z, Hoiem D (2018) Learning without forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence 40(12):2935–2947. https://doi.org/10.1109/TPAMI.2017.2773081. (**Conference name IEEE Transactions on pattern analysis and machine intelligence**)
16. Aljundi R, Belilovsky E, Tuytelaars T, Charlin L, Caccia M, Lin M, Page-Caccia L (2019) Online continual learning with maximal interfered retrieval. In advances in neural information processing systems, vol. 32 (Curran Associates, Inc., 2019)
17. Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM computing surveys (CSUR)
18. Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. In NIPS deep learning and representation learning workshop
19. Vyas A, Jammalamadaka N, Zhu X, Das D, Kaul B, Willke TL (2018) Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In proceedings of the european conference on computer vision (ECCV), pp 550–564
20. Gao Y, Chandra S, Wang Z, Khan L (2018) Adaptive image stream classification via convolutional neural network with intrinsic similarity metrics. In Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining (ACM, London, 2018)
21. Korycki L, Krawczyk B (2021) Class-incremental experience replay for continual learning under concept drift. In 2021 IEEE/CVF Conference on computer vision and pattern recognition workshops (CVPRW) (IEEE, Nashville, TN, USA, 2021), pp 3644–3653. https://doi.org/10.1109/CVPRW53098.2021.00404
22. Din SU, Shao J (2020) Exploiting evolving micro-clusters for data stream classification with emerging class detection. Inf Sci 507:404–420. https://doi.org/10.1016/j.ins.2019.08.050
23. Haque A, Khan L, Baron M (2016) SAND: Semi-Supervised Adaptive Novel Class Detection and Classification over Data Stream. In Thirtieth AAAI conference on artificial intelligence
24. Cai XQ, Zhao P, Ting KM, Mu X, Jiang Y (2019) Nearest neighbor ensembles: an effective method for difficult problems in streaming classification with emerging new classes. In 2019 IEEE international conference on data mining (ICDM), pp 970–975. https://doi.org/10.1109/ICDM.2019.00109. ISSN: 2374-8486
25. Zhang J, Wang T, Ng WWY, Pedrycz W (2022) KNNENS: A k-nearest neighbor ensemble-based method for incremental learning Under data stream with emerging new classes. IEEE Transactions on Neural Networks and Learning Systems pp 1–8. https://doi.org/10.1109/TNNLS.2022.3149991. Conference Name: IEEE Transactions on Neural Networks and Learning Systems
26. Hendrycks D, Gimpel K (2017) A baseline for detecting misclassified and Out-of-distribution examples in neural networks. In 5th international conference in learning representations. Toulon, France
27. Liang S, Li Y, Srikant R (2017) Enhancing the reliability of out-of-distribution image detection in neural networks. In 5th international conference in learning representations. Toulon, France
28. Adimoolam M, Mohan S, J A, Srivastava G (2022) A novel technique to detect and track multiple objects in dynamic video surveillance systems. https://doi.org/10.9781/ijimai.2022.01.002. Accepted: 2022-10-10T11:18:19Z Publisher: International Journal of Interactive Multimedia and Artificial Intelligence (IJIMAI)
29. Chaudhry A, Rohrbach M, Elhoseiny M, Ajanthan T, Dokania PK, Torr PHS, Ranzato M (2019) On tiny episodic memories in continual learning. In 33rd Conference on Neural Information Processing Systems (NeurIPS, Vancouver, Canada, 2019)
30. Hayes TL, Cahill ND, Kanan C (2019) Memory efficient experience replay for streaming learning. In 2019 International Conference on Robotics and Automation (ICRA) (2019), pp 9769–9776. https://doi.org/10.1109/ICRA.2019.8793982. ISSN: 2577-087X
31. Chambers L, Gaber MM, Abdallah ZS (2020) DeepStreamCE: A Streaming Approach to Concept Evolution Detection in Deep Neural Networks. arXiv:2004.04116, [cs, stat]
32. Chambers L (2022) Gaber MM (2022) DeepStreamOS: Fast open-Set classification for convolutional neural networks. Pattern Recogn Lett 154:75–82. https://doi.org/10.1016/j.patrec.2022.01.011
33. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In ICLR 2015 (San Diego, 2015)
34. Krizhevsky A (2009) Learning multiple layers of features from tiny images. University of Toronto, Tech. rep
35. Xiao H, Rasul K, Vollgraf R (2017) Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747
36. Staszewski P, Jaworski M, Cao J, Rutkowski L (2021) A New Approach to Descriptors Generation for Image Retrieval by Analyzing Activations of Deep Neural Network Layers. IEEE Transactions on Neural Networks and Learning Systems pp 1–8. https://doi.org/10.1109/TNNLS.2021.3084633. Conference Name: IEEE Transactions on Neural Networks and Learning Systems
37. Bifet A, Gavaldã R (2009) Adaptive learning from evolving data streams. In Advances in Intelligent Data Analysis VIII, ed. by Adams NM, Robardet C, Siebes A, Boulicaut JF (Springer, Berlin, Heidelberg, 2009), Lecture Notes in Computer Science, pp 249–260. https://doi.org/10.1007/978-3-642-03915-7_22

38. Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In Advances in Artificial Intelligence – SBIA, ed. by Bazzan ALC, Labidi S (Springer, Berlin, Heidelberg, 2004), Lecture Notes in Computer Science, pp 286–295. https://doi.org/10.1007/978-3-540-28645-5_29
39. Castro FM, Marin-Jimenez MJ, Guil N, Schmid C, Alahari K (2018) End-to-end incremental learning. In Proceedings of the European Conference on Computer Vision (ECCV), pp 233–248
40. Spinosa EJ, de Leon AP, de Carvalho F, Gama J (2007) OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams. In Proceedings of the 2007 ACM symposium on Applied computing (Association for Computing Machinery, New York, NY, USA, 2007), SAC '07, pp 448–452. https://doi.org/10.1145/1244002.1244107
41. de Faria ER, de Leon Ponce, Ferreira Carvalho AC, Gama J (2016) MINAS: multiclass learning algorithm for novelty detection in data streams. Data Min Knowl Disc 30(3):640–680. https://doi.org/10.1007/s10618-015-0433-y
42. Masud M, Gao J, Khan L, Han J, Thuraisingham BM (2011) Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints. IEEE Trans Knowl Data Eng 23(6):859–874. https://doi.org/10.1109/TKDE.2010.61
43. Haque A, Khan L, Baron M, Thuraisingham B, Aggarwal C (2016) Efficient handling of concept drift and concept evolution over stream data. In 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp 481–492. https://doi.org/10.1109/ICDE.2016.7498264. 00046 ISSN: null
44. Verleysen M, François D (2005) The curse of dimensionality in data mining and time series prediction. In Computational Intelligence and Bioinspired Systems, ed. by Cabestany J, Prieto A, Sandoval F (Springer, Berlin, Heidelberg, 2005), Lecture Notes in Computer Science, pp 758–770. https://doi.org/10.1007/11494669_93
45. Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G (2019) Learning under Concept Drift: A Review. IEEE Trans Knowl Data Eng 31(12):2346–2363. https://doi.org/10.1109/TKDE.2018.2876857. (**Conference Name: IEEE Transactions on Knowledge and Data Engineering**)
46. Yan MMW (2020) Accurate detecting concept drift in evolving data streams. ICT Express 6(4):332–338. https://doi.org/10.1016/j.icte.2020.05.011
47. Agrahari S, Singh AK (2021) Concept Drift Detection in Data Stream Mining?: A literature review. Journal of King Saud University - Computer and Information Sciences. https://doi.org/10.1016/j.jksuci.2021.11.006
48. Page ES (1954) Continuous Inspection Schemes. Biometrika 41(12):100–115. https://doi.org/10.2307/2333009
49. Roberts SW (2000) Control Chart Tests Based on Geometric Moving Averages. Technometrics 42(1):97–101. https://doi.org/10.1080/00401706.2000.10485986, Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/00401706.2000.10485986
50. Bifet A, GavaldÃ R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM International Conference on Data Mining, Proceedings (Society for Industrial and Applied Mathematics, 2007), pp 443–448. https://doi.org/10.1137/1.9781611972771.42
51. Raab C, Heusinger M, Schleif FM (2020) Reactive Soft Prototype Computing for Concept Drift Streams. Neurocomputing 416:340–351. https://doi.org/10.1016/j.neucom.2019.11.111
52. Frías-Blanco I, Campo-Ávila Jd, Ramos-Jiménez G, Morales-Bueno R, Ortiz-Díaz A, Caballero-Mota Y (2015) Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. IEEE Transactions on Knowledge and Data Engineering 27(3), 810–823 (2015). https://doi.org/10.1109/TKDE.2014.2345382. Conference Name: IEEE Transactions on Knowledge and Data Engineering
53. Baena-García M, Campo-Ãvila J, Fidalgo-Merino R, Bifet A, Gavald R, Morales-Bueno R (2006) Early Drift Detection Method. 4th ECML PKDD international workshop on knowledge discovery
54. Gonçalves PM, de Carvalho Santos SGT, Barros RSM, Vieira DCL (2014) A comparative study on concept drift detectors. Expert Syst Appl 41(18):8144–8156. https://doi.org/10.1016/j.eswa.2014.07.019.00073
55. Ditzler G, Roveri M, Alippi C (2015) Polikar R (2015) Learning in Nonstationary Environments: A Survey. IEEE Comput Intell Mag 10(4):12–25. https://doi.org/10.1109/MCI.2015.2471196.00315
56. Kantchelian A, Afroz S, Huang L, Islam AC, Miller B, Tschantz MC, Greenstadt R, Joseph AD, Tygar JD (2013) Approaches to adversarial drift. In Proceedings of the 2013 ACM workshop on Artificial intelligence and security (Association for Computing Machinery, New York, NY, USA, 2013), AISec '13, pp 99–110. https://doi.org/10.1145/2517312.2517320
57. Ryan S, Corizzo R, Kiringa I, Japkowicz N (2019) Deep learning versus conventional learning in data streams with concept drifts. In 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), pp 1306–1313. https://doi.org/10.1109/ICMLA.2019.00213
58. Disabato S, Roveri M (2019) Learning convolutional neural networks in presence of concept drift. In 2019 International Joint Conference on Neural Networks (IJCNN), pp 1–8. https://doi.org/10.1109/IJCNN.2019.8851731. ISSN: 2161-4407
59. Goodfellow IJ, Mirza M, Xiao D, Courville A, Bengio Y (2015) An empirical investigation of catastrophic forgetting in gradient-based neural networks. In International Conference on Learning Representations (ICLR) 2014 (ICLR, Banff, Canada, 2015)
60. Mensink T, Verbeek J, Perronnin F (2013) Csurka G (2013) Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(11):2624–2637. https://doi.org/10.1109/TPAMI.2013.83. (**IEEE Transactions on Pattern Analysis and Machine Intelligence**)
61. Zhu YN, Li YF (2020) Semi-Supervised Streaming Learning with Emerging New Labels. Proc AAAI Conf Artif Intell 34(04):7015–7022. https://doi.org/10.1609/aaai.v34i04.6186. Number: 04
62. Mu X, Ting KM, Zhou ZH (2017) Classification Under Streaming Emerging New Classes: A Solution Using Completely-Random Trees. IEEE Trans Knowl Data Eng 29(8):1605–1618. https://doi.org/10.1109/TKDE.2017.2691702. (**IEEE Transactions on Components, Packaging and Manufacturing Technology**)
63. Gao Y, Chandra S, Li Y, Khan L (2022) Bhavani T (2022) SACCOS: A Semi-Supervised Framework for Emerging Class Detection and Concept Drift Adaption Over Data Streams. IEEE Trans Knowl Data Eng 34(3):1416–1426. https://doi.org/10.1109/TKDE.2020.2993193. (**IEEE Transactions on Knowledge and Data Engineering**)
64. Mu X, Zhu F, Du J, Lim EP, Zhou ZH (2017) Streaming Classification with Emerging New Class by Class Matrix Sketching. Proceedings of the AAAI Conference on Artificial Intelligence 31(1). https://doi.org/10.1609/aaai.v31i1.10842. Number: 1
65. Ferguson M, Ak R, Lee YTT, Law KH (2017) Automatic localization of casting defects with convolutional neural networks. In 2017 IEEE International Conference on Big Data (Big Data) (2017), pp 1726–1735. https://doi.org/10.1109/BigData.2017.8258115
66. Ong EJ, Husain S, Bober M (2022) Understanding the Distributions of Aggregation Layers in Deep Neural Networks. IEEE Transactions on Neural Networks and Learning Systems pp 1–15 (2022). https://doi.org/10.1109/TNNLS.2022.3207790. Conference Name: IEEE Transactions on Neural Networks and Learning Systems
67. Ulger F, Yuksel SE, Yilmaz A, Gokcen D (2023) Fine-Grained Classification of Solder Joints With \alpha-Skew Jensen-Shannon Divergence. IEEE Transactions on Components, Packaging and

Manufacturing Technology 13(2):257–264. https://doi.org/10.1109/TCPMT.2023.3249193. (**IEEE Transactions on Components, Packaging and Manufacturing Technology**)

68. McInnes L, Healy J, Melville J (2020) UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction
69. Roseberry M, Krawczyk B, Cano A (2019) Multi-Label Punitive kNN with Self-Adjusting Memory for Drifting Data Streams. ACM Trans Knowl Discov Data 13(6):60:1-60:31. https://doi.org/10.1145/3363573
70. Losing V, Hammer B, Wersing H (2018) Tackling heterogeneous concept drift with the Self-Adjusting Memory (SAM). Knowl Inf Syst 54(1):171–201. https://doi.org/10.1007/s10115-017-1137-y
71. Losing V, Hammer B, Wersing H (2016) KNN classifier with self adjusting memory for heterogeneous concept drift. In 2016 IEEE 16th International Conference on Data Mining (ICDM) (2016), pp 291–300. https://doi.org/10.1109/ICDM.2016.0040. ISSN: 2374-8486
72. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chintala S (2019) PyTorch: an imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, vol. 32 (Curran Associates, Inc., 2019)
73. Chaudhry A, Dokania PK, Ajanthan T, Torr PHS (2018) Riemannian walk for incremental learning: understanding forgetting and intransigence. In Proceedings of the European Conference on Computer Vision (ECCV), pp 532–547
74. Vitter JS (1985) Random sampling with a reservoir. ACM Transactions on Mathematical Software 11(1):37–57. https://doi.org/10.1145/3147.3165
75. Mai Z, Kim H, Jeong J, Sanner S (2020) Batch-level experience replay with review for continual learning. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (IEEE, Seattle, USA, 2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.