ODIN: Automated Drift Detection and Recovery in Video Analytics

Abhijit Suprem[1] asuprem@gatech.edu
Joy Arulraj[1] arulraj@gatech.edu
Calton Pu[1] calton.pu@cc.gatech.edu
Joao Ferreira[2] jef@ime.usp.br
[1]Georgia Institute of Technology, [2]University of Sao Paulo

ABSTRACT

Recent advances in computer vision have led to a resurgence of interest in visual data analytics. Researchers are developing systems for effectively and efficiently analyzing visual data at scale. A significant challenge that these systems encounter lies in the drift in real-world visual data. For instance, a model for self-driving vehicles that is not trained on images containing snow does not work well when it encounters them in practice. This drift phenomenon limits the accuracy of models employed for visual data analytics.

In this paper, we present a visual data analytics system, called ODIN, that automatically detects and recovers from drift. ODIN uses adversarial autoencoders to learn the distribution of high-dimensional images. We present an unsupervised algorithm for detecting drift by comparing the distributions of the given data against that of previously seen data. When ODIN detects drift, it invokes a drift recovery algorithm to deploy specialized models tailored towards the novel data points. These specialized models outperform their non-specialized counterpart on accuracy, performance, and memory footprint. Lastly, we present a model selection algorithm for picking an ensemble of best-fit specialized models to process a given input. We evaluate the efficacy and efficiency of ODIN on high-resolution dashboard camera videos captured under diverse environments from the Berkeley DeepDrive dataset. We demonstrate that ODIN's models deliver 6× higher throughput, 2× higher accuracy, and 6× smaller memory footprint compared to a baseline system without automated drift detection and recovery.

PVLDB Reference Format:
Abhijit Suprem, Joy Arulraj, Calton Pu, Joao Ferreira. ODIN: Automated Drift Detection and Recovery in Video Analytics. *PVLDB*, 13(11): 2453-2465, 2020.
DOI: https://doi.org/10.14778/3407790.3407837

1. INTRODUCTION

Recent advances in computer vision (*e.g.*, image classification [19], object detection [26], and object tracking [16]) have led to a resurgence of interest in visual data analytics. Researchers are developing database management systems (DBMSs) for analyzing visual data at scale [15, 11, 17, 14]. While these systems deliver high performance, they suffer from a major limitation that constrains their accuracy on real-world visual data. They assume that all the frames of videos stem from a static distribution. In practice, the visual data *drifts* over time because it comes from a dynamic, time-evolving distribution. For instance, a machine learning (ML) model for self-driving vehicles that is not trained on images containing snow does not work well when it encounters them in practice [38]. This phenomenon is referred to as *concept drift* [6, 34], and it limits the efficacy of ML models employed in visual DBMSs.

Challenges. Concept drift is well studied in the domain of low-dimensional, structured data analysis [6]. For instance, Kalman filtering is a widely-used technique for recovering from data drift due to sensor failures [12]. However, these techniques cannot cope with drift in high-dimensional, unstructured data (*e.g.*, images [1], videos [32]). State-of-the-art ML models assume that the data comes from a static distribution. This *closed-world assumption* does not hold in real-world settings where data is continuously drifting [34]. Consider an image classification task. These models assume that: (1) the data space is known a priori (*i.e.*, the list of classes is well defined during training), and (2) that the training data is representative of the test data. These assumptions are invalid in practice due to drift. This reduces the detection accuracy of these models when drift occurs and the distribution of the input data changes.

Prior Work. Recently, researchers have highlighted the challenges associated with coping with drift [15, 14]. To detect and recover from drift, the authors recommend that the user manually identify the evolution of the input distribution and construct models specialized for the novel data points (*i.e.*, outliers). The DBMS then selects the appropriate user-constructed model based on query-specific accuracy and performance constraints [15]. For instance, in case of a traffic surveillance dataset, it uses an expensive, more accurate model for object detection under high traffic conditions and a slower, less accurate model otherwise [14]. The key limitation of this approach is that it is not automated. This delays the drift detection and recovery processes, thereby degrading the accuracy and performance of the DBMS.

Our Approach. In this paper, we present a visual DBMS, called ODIN, that automatically detects and recovers from drift. We present an unsupervised algorithm that identifies outliers by learning the input distribution using adversarial autoencoders. ODIN's DETECTOR relies on a distance metric based on generative adversarial networks. We show that this distance metric outperforms state-of-the-art outlier detection algorithms on high-dimensional visual data (since existing algorithms are tailored for low-dimensional structured data). Using DETECTOR, ODIN automatically differentiates between key concepts in the dataset (*e.g.*, weather conditions or time-of-day). After detecting drift, ODIN's SPECIALIZER constructs a family of models specialized for the novel data points to recover from the

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407837

changes in input distribution. We show that the specialized models outperform their non-specialized counterparts in both accuracy and performance. We demonstrate that these specialized models are resilient to drift unlike other forms of model specialization (*e.g.*, student models[37]). Lastly, ODIN's SELECTOR picks an ensemble of specialized models from its family of models for processing a given input. We compare the efficacy of several model selection policies for drift recovery. We demonstrate the end-to-end efficacy and efficiency of ODIN on high-resolution dashboard camera videos captured under diverse environments from the Berkeley DeepDrive BDD dataset [38].

Contributions. We make the following contributions:

- We present an unsupervised algorithm for drift detection that learns the input distribution using adversarial autoencoders. We propose a novel distance metric based on generative adversarial networks that is tailored for high-dimensional visual data (§4).
- We introduce a technique for drift recovery using specialized models that are resilient to drift. We present a set of policies for selecting an ensemble of specialized models for processing a given input (§5).
- We implemented our drift detection and recovery algorithms in ODIN and evaluated its efficacy and efficiency on three datasets.
- We demonstrate that ODIN delivers 6× higher throughput and 2× higher object detection accuracy than a static system without automated drift detection and recovery. We show that ODIN delivers 1.5× higher query accuracy than its static counterpart on canonical aggregation query over visual data (§6).

2. BACKGROUND

We begin by motivating the need for detecting and recovering from drift in §2.1. We then present an overview of concept drift to better appreciate the drift detection and recovery algorithms in §2.2. Lastly, we describe the generative models that ODIN uses in §2.3.

2.1 Motivating Example

In this example, we illustrate the benefits of detecting and recovering from drift by constructing specialized models for novel data points. We compare ODIN against a static system with drift detection and recovery disabled on the BDD dataset. This dataset consists of high-resolution, colored images obtained from dashboard camera videos under diverse weather conditions [38]. We examine how a system trained on RAIN-DATA, a cluster in BDD containing videos from overcast and rainy days, performs on DAY-DATA, another cluster in BDD containing videos from clear, sunny days. We defer a detailed description of our experimental setup to §6.1.

ODIN uses two smaller and faster models specialized for RAIN-DATA and DAY-DATA clusters. In contrast, the static system is a single heavyweight YOLO [30] model that is trained on RAIN-DATA. We compare the efficacy and efficiency of the static model against the specialized models that are dynamically constructed by ODIN after it detects drift. The results are shown in Figure 1. We compare four metrics:

- **Detection accuracy:** Accuracy of the object detection model.
- **Query accuracy:** Accuracy of the output of an aggregation query counting the number of cars in the videos.
- **Throughput:** Number of images processed per second (FPS).
- **Memory footprint:** GPU memory occupied by the systems.

ODIN delivers higher detection and query accuracy than the static system by leveraging the specialized models for object detection. The static model trained on the RAIN-DATA subset of BDD has lower accuracy when the data changes to DAY-DATA. ODIN automatically detects this drift in the input data and recovers by deploying a specialized model for DAY-DATA. So, it maintains higher accuracy even in the presence of drift. Furthermore, the smaller, specialized models constructed by ODIN are 6× faster and 6× smaller than the heavyweight model used in the static system. We defer a detailed description of the specialized models to §6.3. This example illustrates the importance of detecting and recovering from drift.

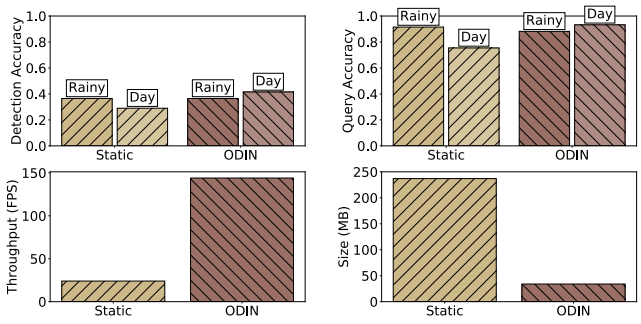


Figure 1: Motivating Example: We compare ODIN against a static system without automated drift detection and recovery on the BDD dataset.

2.2 Concept Drift

Concept drift consists of learning in a non-stationary environment, in which the underlying data distribution (*i.e.*, the joint distribution of the input data and labels P(X,Y)) evolves over time [6, 34]. It is also referred to as *domain adaptation*. We may classify the changes in the data distribution into two categories: (1) task drift, and (2) domain drift [20]. The key distinction between task and domain drift is that the *real* decision boundary only changes under task drift.

Task drift reflects real changes in the world. Formally, this corresponds to the drift in the conditional distribution of labels given the input data (*i.e.*, P(Y | X)), often resulting from an updated definition of the task necessitating a change in the predictive function from the input space to label space.

Domain drift does not occur in reality but rather occurs in the ML model reflecting this reality. In practice, this type of drift arises when the model does not identify all the relevant features or cannot cope with class imbalance. Formally, this corresponds to the drift in the marginal distribution of the input data (*i.e.*, P(X)), with an additional assumption that P(Y | X) remains the same.

ODIN only copes with domain drift. It measures changes in the marginal distribution of the input data (*i.e.*, P(X)). ODIN uses generative models to construct a low-dimensional projection of the given images and then clusters the projected images. We will next provide an overview of generative models.

2.3 Generative Models

Generative models are a category of neural networks for synthesizing new data points that appear as if they are drawn from the training data distribution [9]. ODIN uses two types of generative models: (1) autoencoders (AE), and (2) generative adversarial network (GAN). Both approaches project an input a low dimensional space by compressing it.

ODIN uses these low-dimensional projections to detect drift by measuring the distance between existing and novel data points. Intuitively, GANs and AEs try to capture the most important attributes of images during compression, because during training, they must be able to reconstruct an image from a compressed representation. Because these low dimensional representations already capture the

(a) Standard AE **(b)** Adversarial AE **(c)** DA-GAN

Figure 2: Latent spaces: The latent spaces provide crucial clues. The standard AE's latent space has holes, indicating unsuitability for drift detection. The adversarial AE's latent space is smooth; the blurriness indicates some loss of information. The DA-GAN's latent space is smooth with better reconstruction, indicating most of the underlying distribution has been captured in the latent space.

important attributes, *i.e.* the underlying distribution, it is easier to detect changes in the underlying distribution in this space.

Standard Autoencoder. A standard autoencoder (AE) consists of an encoder and a decoder in series. In an AE:

- The encoder E compresses by mapping an input image x of n dims to a latent space of z dims, where $n >> z$.
- The decoder G takes $z = E(x)$ as input and reconstructs x. We refer to this reconstruction as $x' = G(E(x))$.

An AE is trained using the reconstruction loss (binary cross-entropy loss in Equation 5). AEs display an *irregular mapping* problem because of nonlinear activations [39]. It can project an input to any random point in the latent space $\Re^z$; this creates holes in the latent space, shown in Figure 2a. These holes are regions that the decoder cannot reconstruct. When the underlying distribution changes due to drift, the AE can project these new inputs into the holes, leading to empty or invalid reconstructions by the decoder.

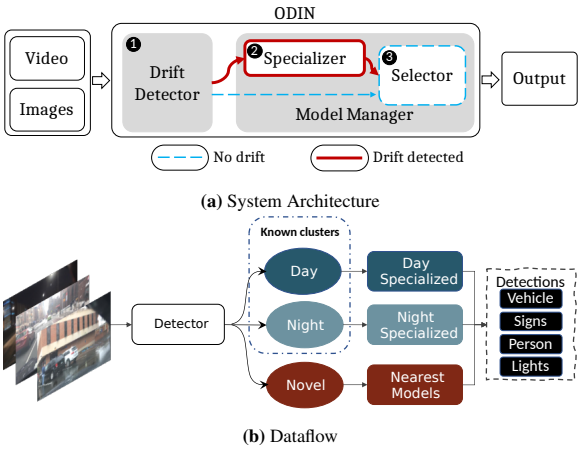
Adversarial Autoencoder. An adversarial AE closes the holes of the standard AE latent space by enforcing a smoothness constraint [25] that ensures similar data is projected close together. This constraint is enforced using a discriminative network D_Z. D_Z takes two inputs: (1) points drawn from the latent space, and (2) points drawn from a smooth distribution (*e.g.* normal distribution). It is trained to distinguish between these two distributions using a binary cross-entropy loss. In an adversarial AE:

- The encoder E maps the input x to a low-dimensional embedding z. Next, D_Z predicts whether z is drawn from the encoded distribution or the desired distribution.
- The decoder G generates x' from z. The reconstruction loss between x and x' is used to concurrently train both E and G.

With the competition between E and D_Z, the encoder learns to map points to the desired distribution, creating a latent space without holes (Figure 2b). However, the adversarial AE loses some image information, resulting in blurriness.

Generative Adversarial Network (GAN). A GAN consists of two networks in series: (1) a generator network $G(z)$, and (2) a discriminator network $D_I(x)$. $G(z)$ is similar to the decoder in an AE. $D_I(x)$ is similar to the $D_Z(z)$ in an adversarial AE. A GAN uses $D_I(x)$ to improve the quality of the generator. Given a point z in the latent space, $G(z)$ generates an image x'. Then $D_I(x)$ distinguishes between a real image x and a generated image x'.

Dual Adversarial GAN (DA-GAN). In this paper, we present a novel network that combines the modeling capabilities of both adversarial AE and GAN. It consists of four components: (1) an encoder, (2) a decoder, (3) a latent discriminator, and (4) an image discriminator. The decoder of the adversarial AE serves as the generator of the GAN. The latent and image discriminator together improve the latent space (Figure 2c): the latent discriminator makes it smooth, and the image discriminator ensure miinimal information loss by forcing better reconstruction. We call this network a *dual-adversarial GAN* because it contains two discriminators. We defer a detailed description of DA-GAN to §4.3.



(a) System Architecture

(b) Dataflow

Figure 3: Architecture and Dataflow of ODIN. ODIN takes in a sequence of images as input. ❶ DETECTOR uses a DA-GAN to obtain the low-dimensional latent projection of the input and to identify new clusters without supervision. ❷ If drift is detected, SPECIALIZER generates a specialized model for the newly detected cluster. ❸ Lastly, SELECTOR chooses the appropriate specialized model for the given input.

3. SYSTEM OVERVIEW

We now present an overview of the architecture of ODIN. As illustrated in Figure 3a, ODIN consists of three components:

❶ DETECTOR identifies drift in the given data using an unsupervised algorithm tailored for high-dimensional data. It learns the distribution of clustered *density bands* in the given data. We present this component in §4. Intuitively, a high-density region in the latent space represents a latent concept and changes in this region indicate changes in the concept itself (*i.e.*, concept drift). A key component of DETECTOR is the distance metric that it employs for clustering data points into density bands in an unsupervised manner. We show that distance metrics employed for structured data do not work well with high-dimensional visual data. We make the case for modeling the latent space in visual data using a DA-GAN and using its latent space density to detect drift (§4.3).

❷ When DETECTOR identifies drift, ODIN relies on the SPECIALIZER to recover from the detected drift by generating specialized models for newly detected clusters. SPECIALIZER allows ODIN to deliver high accuracy across all clusters. We present this component in §5. We illustrate the importance of specialization by comparing the accuracy of a non-specialized model trained on the entire dataset to specialized models optimized for particular clusters.

❸ Lastly, the SELECTOR is responsible for choosing the appropriate specialized model for a given input to perform inference. When drift occurs, the SPECIALIZER may take time to collect sufficient novel data points before constructing a model for the newly detected cluster. During this phase, SELECTOR dynamically creates an ensemble of specialized models from nearby clusters for inference. We present this component in §5.3. We consider SPECIALIZER and SELECTOR to be a part of the MODELMANAGER within ODIN.

MODELMANAGER is responsible for generating specialized models and choosing the appropriate one during inference.

Dataflow. Figure 3b illustrates the flow of data in ODIN. Given an image, DETECTOR performs dimensionality reduction to get its lower-dimensional manifold. It uses this manifold to map it to existing clusters from previously seen data. If the input belongs to an existing cluster, SELECTOR picks the associated model for inference (*e.g.*, identifying objects in the given BDD image). If that is not the case, then it picks an ensemble of specialized models from nearby clusters for inference. Simultaneously, SPECIALIZER records the input to train a specialized model.

4. DRIFT DETECTION

In this section, we present the unsupervised algorithm for drift detection technique that ODIN employs.

Overview. DETECTOR performs the following tasks. ❶ It first learns a low-dimensional representation of visual data using DA-GAN (§4.3). ❷ It then uses this low-dimensional representation to cluster the data points using without being affected by the curse of dimensionality[1]. ❸ It next constructs a succinct topological representation of these clusters using *density bands* [13] (§4.1). This improved representation only captures the *high-density* region of the cluster, where most of the points exist. DETECTOR learns the distribution parameters of the density band associated with each cluster. ❹ Finally, it detects drift by comparing the distribution of novel data points against that of existing clusters using their KL divergence (§4.1).

In the rest of this section, we first formalize the notion of density bands in §4.1. We then illustrate the challenges associated with using AEs to detect drift in §4.2. We then present how DETECTOR leverages DA-GAN as a distance-preserving dimensionality reduction technique in high-dimensional spaces in §4.3. We next discuss how we train DA-GAN in §4.4. Lastly, we describe how DETECTOR detects drift by comparing the distributions using KL divergence in §4.5.

4.1 Density Bands

Consider a data space D. Let D_k denote the set of points in a cluster k associated with a particular concept. A high-density band in D_k is a subset of D_k that contains more than 50% of the points in that cluster.

To obtain a density band, DETECTOR first estimates the distribution of points in D_k. It centers the band at the distribution peak of the cluster (*i.e.*, where most points are present with respect to the cluster's center. It then expands the band inwards to the center and outwards to the cluster edges to compute the lower and upper bounds of the density band. For each cluster, DETECTOR uses a pre-defined threshold on the fraction of points that must be present within the band to determine its bounds; we use $\Delta = 0.5$. Figure 4 illustrates this technique for constructing bands.

For a given cluster D_k with p data points, DETECTOR computes its centroid $D_k^C = (\sum_{i=1}^{p} x_i)/p$ and its probability mass function. Let $f_\Delta(x)$ be a continuous density function of D_k that is estimated on a normalized distance metric $d : \Re^n \rightarrow [0, 1]$. Here, d measures the distance between any point x_i and the centroid D_k^C. Thus, $f_\Delta(x)$ captures the distribution of D_k's points with respect to the centroid D_k^C. DETECTOR then uses $f_\Delta(x)$ to compute the density band. A density band Δ is defined by two bounds: $[\Delta_l, \Delta_h]$, where $0 \leq \Delta_l < \Delta_h \leq 1$. As shown in Figure 4, Δ represents the fraction of points in the cluster within the lower and upper bounds Δ_l and Δ_h, respectively. DETECTOR computes the density band's parameters based on Δ and its density function:

$$\int_{\Delta_l}^{\Delta_h} f_\Delta(x)dx = \Delta \quad (1)$$

[1]This phenomena refers to the differencces in classifying high dimensional data vs. low dimensional data. Distance metrics tend towards 0 in high dimensions.

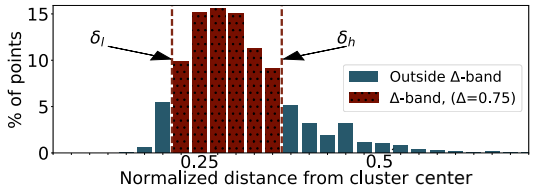


Figure 4: Visualization of Δ-band: Histogram of embedded points in a cluster The hypersphere region up to radius ∼0.18 is empty. The high-density band, with $\Delta = 0.75$, is highlighted, with its bounds Δ_l and Δ_h.

KL Divergence. While processing a given data point, DETECTOR maps it to existing permanent clusters associated with known concepts or to a single temporary cluster. We defer a detailed description of this algorithm to §5.1. A point that falls inside a permanent cluster's Δ-band is assigned to that cluster. A point that falls outside all of the permanent clusters' Δ-bands is assigned to the temporary cluster. DETECTOR continuously updates the parameters of the temporary cluster's Δ-band based on the new data points in the input stream. It detects drift by using KL divergence to compare the posterior distribution of the temporary cluster's Δ-band after a point is added against the prior distribution before a point is added. The KL divergence between the two distributions modeling a data point x (*i.e.*, the prior P_A and the posterior P_B) is given by:

$$D_{KL}(P_A||P_B) = -\sum_{x \in X} P_A(x) \log(P_B(x)/P_A(x)) \quad (2)$$

Here P_A is the *expected* prior and P_B is the *live* posterior observed in practice. When the distribution inside the Δ-band before and after a point is added no longer changes ($D_{KL} \rightarrow 0$ when $P_B = P_A$), ODIN consider the temporary cluster as stable. This indicates the presence of drift, since there are enough points in the temporary cluster to indicate the introduction of a new concept (*e.g.*, snowy images). ODIN converts the temporary cluster to a permanent cluster and adds it to the set of permanent clusters (§5.3). It concurrently constructs a specialized model for this cluster (§5.1). Lastly, it initializes a new empty temporary cluster for processing subsequent points.

Manifold Learning. With KL divergence, DETECTOR measures the changes in the input distribution. However, it still needs a suitable distance metric for modeling the distribution of the input and for projecting it from high-dimensional images to a lower-dimensional manifold. We next illustrate the challenges associated with detecting drift in high-dimensional visual data.

4.2 Drift Detection in Images

Real-world datasets often fit a manifold whose dimensionality is lower than the raw data. Consider the digit-classification task on the MNIST dataset [4]. We may project the 28×28 images in this dataset (*i.e.*, 784 dimensions) onto a ten-dimensional manifold of digits using a neural network with one-hot encoded outputs. Here the network $N_{MNIST} : \Re^{784} \rightarrow \Re^{10}$ learns to approximate the projection from the raw data to the manifold. While this network works well in the absence of concept drift, it will start to misclassify novel data points in the presence of drift. This is because the changes in the data distribution necessitate a shift in the projection as well.

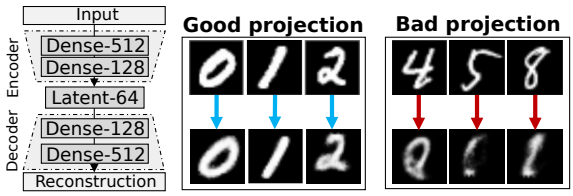


Figure 5: Projection Failure: We use two similar datasets to demonstrate the projection failure in the presence of concept drift. A model trained on a subset of MNIST (digits `0-2`) fails to reconstruct outliers (digits `3-9`).

Figure 5 illustrates this problem of a *projection failure*. We train an AE with four dense layers each with ReLU activation. The dimensionality of the latent space of this AE is 64. We train it on a subset of MNIST containing three digits `0-2` and then test it on the entire dataset. We observe that the AE cannot reconstruct digits `3-9`, since it expects the images to be drawn from a distribution comprising of digits `0-2`. Even though the images are visually similar (*i.e.*, 28×28 black-and-white images of digits), the concept drift in the testing inputs causes the reconstruction to fail. This is because the AE only learns the projection of digits `0-2`, instead of learning the projection of black-and-white images in general.

The most notable observation from this experiment is that high reconstruction error of the AE indicates drift. This means the latent space for drifted data is far from the latent space of known data, since the autoencoder only learns the projection over the known data. Since the latent space is at a lower dimensionality than the input images, measuring drift there bypasses the curse of dimensionality and can be more effective. So, DETECTOR measures drift using the latent space representation. However, AEs have problems in latent space representation, such as holes(Figure 2a). We next present a distance-preserving dimensionality reduction technique that works well on high-dimensional images and avoids problems of AEs.

4.3 Dual-Adversarial GAN

DETECTOR computes Δ-bands and KL divergence on this latent space. As we discussed in the overview of generative models (§2.3): (1) AEs create holes in the latent space during projection, (2) Adversarial AEs lose some information in the image while constructing smoother projections, and (3) GANs are designed for image synthesis, not representation modeling.

Overview. We present a network, called Dual-Adversarial GAN (DA-GAN), that combines an adversarial AE and a GAN to exploit their latent encoding and image information preserving properties, respectively. We use this network to map images to a low-dimensional latent space. The adversarial AE ensures that the latent space matches the desired smooth distribution (*e.g.*, normal distribution). The GAN ensures that the latent space does not lose important information during encoding by focusing on image reconstruction. Since the latent discriminator is trained on the desired smooth distribution, it is adept at discriminating the inlier frames from the outlier frames which should be mapped to a different distribution [36]. In this manner, DA-GAN functions as a distance-preserving projection technique that works well on high-dimensional data.

Structure. The structure of the DA-GAN is shown in Figure 6. It consists of four components: ❶ Encoder $E(x)$, ❷ Decoder $G(z)$, ❸ Latent discriminator $D_Z(z)$, and ❹ Image discriminator $D_I(x)$. We keep the basic structure of an autoencoder and a GAN intact by using an encoder and a decoder . The encoder maps an input x to the latent space: $z = E(x)$ The decoder seeks to reconstruct x using z: $x' = G(z)$. We introduce two adversarial discriminators.

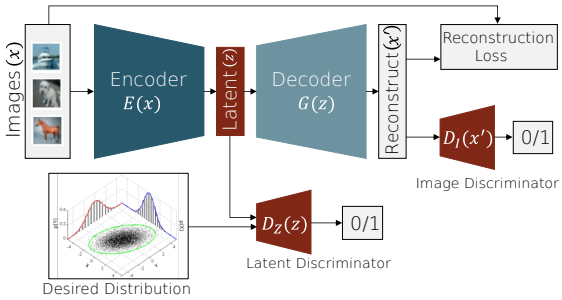


Figure 6: Dual-Adversarial GAN: The dual adversarial GAN has three loss functions: (1) latent discriminator loss ($\mathcal{L}_Z$), (2) image discriminator loss ($\mathcal{L}_I$), and (3) reconstruction loss ($\mathcal{L}_R$).

❸ The first discriminator $D_Z(z)$ imposes a prior on the latent space z. $D_Z(z)$ learns to minimize a binary cross-entropy loss $\mathcal{L}_Z$ between points drawn from the normal distribution $\mathcal{N}(0,1)$ and from the encoded latent space $D_Z(E(x))$:

$$\mathcal{L}_Z = \log(D_Z(\mathcal{N}(0,1))) + \log(1 - D_Z(E(x))) \quad (3)$$

$D_Z(z)$ forces the encoder $E(x)$ to learn clearer separations between classes and to create better reconstructions.

❹ The second discriminator $D_I(x)$ counters the blurriness caused by loss of information in the latent space [24]. This adversarial image discriminator operates on the output of the decoder $G(z)$ (*i.e.*, x'). It learns to minimize $\mathcal{L}_I$ in Equation 4. $\mathcal{L}_I$ is a binary cross-entropy loss to compare the true image x and a reconstruction from a random point in the normal distribution $G(\mathcal{N}(0,1))$.

$$\mathcal{L}_I = \log(D_I(x)) + \log(1 - D_I(G(\mathcal{N}(0,1)))) \quad (4)$$

$D_I(x)$ reduces information loss by forcing $E(x)$ to encode more useful information, allowing $G(z)$ to create better reconstructions.

Lastly, we use the pixel-wise reconstruction loss $\mathcal{L}_R$ in Equation 5 to compare the input image x to the output image $x' = G(E(x))$.

$$\mathcal{L}_R = -E_z[\log(x'|x)] \quad (5)$$

The two adversarial losses $\mathcal{L}_Z$ and $\mathcal{L}_I$ impose the dual constraints of: (1) smoother latent space without holes, and (2) high quality encoding with minimal loss of information, respectively.

Construction. Figure 7 illustrates the components of DA-GAN. The encoder consists of Resnet blocks[10]. We pool the final features by channel to extract the distribution features representing the input. During tranining, these features are passed to the latent discriminator. The distribution features are also passed through deconvolutional Resnet blocks to reconstruct the original input, and the reconstructed image is passed through the image discriminator.

Comparison to U-NET. U-NET is a neural network that performs residual transfer from encoder to decoder by bypassing the latent space [5], allowing U-NET to learn to better reconstruct the image. However, bypassing the latent space skips information from being encoded in the latent space. This would prevent drift detection, since the underlying distribution is not properly encoded.

4.4 DA-GAN Training

We partition each dataset into two sets of classes: (1) *known classes* for training the DETECTOR, and (2) *unknown classes* for testing the DETECTOR. The training procedure ensures that DA-GAN learns to model the distribution of the known classes. During testing, we present data points from both known and unknown classes to evaluate the ability of the DETECTOR to detect drift.

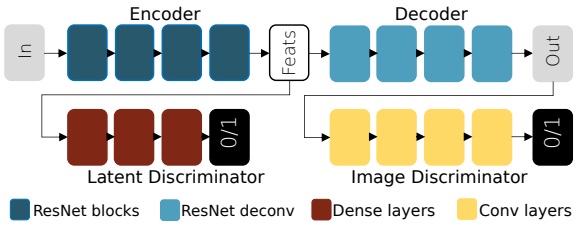


Figure 7: DA-GAN details: The blocks of the DA-GAN model. The encoder and decoder are derived from Resnet [10]. The output of the encoder consists of 1024 features. We illustrate the adversarial discriminators that contribute to $\mathcal{L}_Z$ and $\mathcal{L}_I$.

Loss Functions. The overall loss function for the GAN is shown in Equation 6. We compute the weighted sum of three loss functions: (1) the latent discriminator loss ($\mathcal{L}_Z$), (2) the image discriminator loss ($\mathcal{L}_I$), and (3) the standard reconstruction loss ($\mathcal{L}_R$).

$$\mathcal{L} = \lambda_Z \mathcal{L}_Z + \lambda_I \mathcal{L}_I + \lambda_R \mathcal{L}_R \tag{6}$$

Since the discriminators are adversarial, they must be equally weighted. If λ_Z and λ_I differ, then the learning landscape is more difficult and render the training procedure to be unstable. Thus, we let $\lambda_Z = \lambda_I = 1$.

The reconstruction loss λ_R ensures the encoder is encoding enough information in the latent space for the decoder. We do not require the synthetic images created by the decoder. We only need the decoder to be good at reconstructing the input, since that demonstrates that the encoder is recording relevant information in the latent space. However, λ_R also creates the holes in the latent space, as shown in Figure 2a. We ensure that the latent discriminator loss is prioritized over λ_R by setting $\lambda_R = 0.5\lambda_Z = 0.5$, which closes the latent space holes.

Training. We use the adversarial training procedure shown in Algorithm 1. In each iteration, the components of DA-GAN are updated sequentially. The image discriminator is trained to distinguish between real images and synthetic images generated by the decoder (Lines 5-7). The decoder is trained to fool the image discriminator so that it mistakes synthetic images for real images (Line 8). The latent discriminator is trained to distinguish between points drawn from a normal distribution and points generated by the encoder (Lines 9-11). The encoder is trained to fool the latent discriminator so that it mistakes the points generated by the encoder for points coming from a normal distribution (Line 12). The encoder will succeed only if it maps the input images to the desired smooth distribution. Finally, the autoencoder is updated to minimize the pixel-wise reconstruction loss (Lines 13). This training procedure allows DA-GAN to deliver high image fidelity, since both encoder and decoder must work together to reconstruct the input.

4.5 Clustering

Lastly, we describe how DETECTOR detects drift using DA-GAN (§4.3) and Δ-bands with KL divergence (§4.1). DETECTOR first projects images to a lower dimensional manifold using DA-GAN. After training DA-GAN, DETECTOR only uses the encoder for projecting images.

While processing a stream of incoming data points, DETECTOR maintains a collection of clusters. For a given point, DETECTOR first projects it to a low-dimensional manifold using DA-GAN's encoder. ❶ If the point falls within the Δ-band of an existing cluster, it is added to that cluster. DETECTOR updates that cluster's Δ-band using Equation 1. ❷ If the point falls outside all of the existing Δ-bands, then DETECTOR adds it to a temporary cluster. It

Algorithm 1: DA-GAN Training Iteration

input : Encoder `E()`, Decoder `G()`, Latent Discriminator $\texttt{D}_\texttt{Z}()$, Image Discriminator $\texttt{D}_\texttt{I}()$
output : Trained DA-GAN
functions : `BCE(a,b)`: Binary cross entropy loss between a, b; `GetRandomNormal()`: Sample numbers from $\mathcal{N}(0,1)$; `GetRandomBatch()`: Sample images from BDD; `Backpropagate(a)`: Backpropagate loss a over DA-GAN

```
// Set up targets
1  y_real, y_fake = ones(), zeros()
2  z_real, z_fake = ones(), zeros()
   // Get Minibatches
3  z' ← GetRandomNormal();   x ← GetRandomBatch()
4  x' = G(z');   z = E(x)
   // Update the Image Discriminator
5  LossReal_DI = BCE(DI(x), y_real)
6  LossFake_DI = BCE(DI(x'), y_fake)
7  Backpropagate(LossReal_DI + LossFake_DI)
   // Update the Decoder
8  Backpropagate(BCE(D_I(x'), y_real))
   // Update the Latent Discriminator
9  LossReal_DZ = BCE(DZ(z'), z_real)
10 LossFake_DZ = BCE(DZ(z), z_fake)
11 Backpropagate(LossReal_DZ + LossFake_DZ)
   // Update the Encoder
12 Backpropagate(BCE(DZ(z), z_real))
   // Update both Encoder and Decoder
13 Backpropagate(0.5 · BCE(x, x'))
```

recomputes the temporary cluster's Δ-band and distribution. When that Δ-band's upper and lower bounds no longer change and the distribution stabilizes as per Equation 2, DETECTOR converts the temporary cluster to a permanent cluster and adds it to its collection of clusters. It then initializes a new empty temporary cluster to process subsequent points. The addition of a cluster indicates drift (*i.e.*, the discovery of a new region in the input data space).

All of the components of DETECTOR work in tandem to circumvent the curse of dimensionality (§4). For instance, BDD's 1280×720 colored camera images contain ~921K dimensions. DA-GAN's encoder projects these high-dimensional images down to 1024 dimensions (Figure 7) while generating clusters. DETECTOR then maps these clusters to Δ-bands with four dimensions: (1) lower bound Δ_l, (2) upper bound Δ_h, (3) prior P_A, and (4) posterior P_B. Lastly, it detects drift using these Δ-bands and KL divergence. In this manner, we reduce the dimensionality of the drift detection problem from ~921K dimensions to four dimensions. We demonstrate the efficacy of ODIN's drift detector on diverse datasets in §6.2.

5. DRIFT RECOVERY

In this section, we discuss how ODIN recovers from drift. When DETECTOR creates a new cluster after identifying drift, SPECIALIZER generates a new model that is tailored for the newly detected cluster (§5.1). We illustrate the types of models that SPECIALIZER constructs through a case study on object detection (§5.3). Lastly, after constructing the models, MODELMANAGER uses SELECTOR to pick the best-fit specialized models for prediction (§5.3).

5.1 Model Specialization

A model $\mathcal{M}_k$ tailored for a cluster D_k learns a mapping from that cluster's data points to labels $\mathcal{Y}$. MODELMANAGER maintains a collection of models:

$$\{\mathcal{M}\}^n : \{D\}^n \rightarrow \mathcal{Y} \tag{7}$$

Algorithm 2: Model Specialization

input : n models $\{\mathcal{M}\}^n$, data point x_i, computer vision task $\mathcal{T}$
output : $\mathcal{T}(x_i)$, updated models and clusters if needed
parameter : d (DA-GAN distance metric)

```
1  cluster_found = False
2  for M_j ∈ {M}^n do
       // Distance to centroid using DA-GAN
3      d'_{x_i} = d(x_i, M_j^{centroid})
       // Check if inside Δ-band of M_j
4      if Δ_l^j < d'_{x_i} < Δ_h^j then
           // Add point to model's data D_j
5          D_j = D_j ∪ x_i
           // Update the parameters
6          UpdateDeltaBand(D_j); UpdateModel(M_j)
           // Flag for found cluster
7          cluster_found = True
8      end
9  end
10 if cluster_found = False then
11     D_G = D_G ∪ x_i
       // Update the distributions
12     UpdateDeltaBand(D_G)
13     if StableDistribution(D_G) then
14         M_{n+1} ← GenerateNewModel(D_G)
15         D_{n+1} ← GenerateNewCluster(D_G)
16     end
17 end
```

Here, n represents the currently materialized set of models. When DETECTOR identifies a new cluster D_{k+1}, SPECIALIZER constructs a model $\mathcal{M}_{k+1}$ optimized for the points in D_{k+1} and adds it to the set of materialized models. DETECTOR typically maps most of the data points to existing clusters. For these inliers, ODIN only updates their corresponding model with the new data in the associated cluster. SPECIALIZER constructs new models only to cope with the outliers.

Model Generation. Algorithm 2 presents the algorithm for generating models. ODIN uses the distance metric (in this case, DA-GAN) to determine whether specialization is necessary. For each input x_i, the DA-GAN projects it to the latent space. For each existing cluster generated by DETECTOR, SPECIALIZER checks if x_i belongs to that cluster by comparing the distance between the projected x_i and the center of the cluster (Line 3) against the lower and upper bounds of that cluster's Δ-band (Line 4). If the point exists in a cluster, SPECIALIZER updates the cluster's distribution parameters and model using x_i (Lines 5-6; lower dashed line in ODIN's system design in Figure 3a). If x_i belongs in no existing cluster, then ODIN uses DETECTOR to add it to the temporary cluster D_G (Line 11). Under the gradual drift assumption, D_G grows over time as new outliers are added. When D_G's Δ-band no longer exhibits changes (Line 13), SPECIALIZER constructs a new model trained on D_G (Line 14) and DETECTOR creates a new cluster (Line 15).

5.2 Types of Specialized Models

ODIN adopts two approaches to model specialization:

- **Specialized models for improved accuracy**: Upon discovering a new cluster D_k in the data space, SPECIALIZER generates a *specialized* model $\mathcal{M}_k$ to perform the given task on D_k.
- **Lite models for improved performance**: SPECIALIZER uses a student-teacher approach to train a faster, weaker student model(called YOLO-LITE) using the outputs of the slower parent model [37].

Specialized vs Lite Models. Lite models sacrifice accuracy to enable faster training and subsequent deployment. This is because, unlike specialized models, they do not require *oracle labels* from humans or weakly-supervised agents [29] during training. They instead leverage the outputs of an existing parent model [37]. Thus, ODIN trains and deploys a lite model as soon as it detects a new cluster. Later, when the oracle labels for the newly detected cluster is available, SPECIALIZER trains a specialized model and replaces the lite model with its specialized counterpart. We examine the efficacy and efficiency of these two types of models in §6.3.

Case Study: Object Detection. We next illustrate how ODIN specializes models for recovering from drift through a case study on object detection using the YOLO model [30].

YOLOv3. ODIN uses the YOLO object detection model as the baseline object detector ($\mathcal{M}$). YOLO is an efficient detector that performs inferences in a single pass over the images. It generates region proposals and combines them with a classification model to concurrently segment images and perform dense object labeling.

The YOLO network consists of 24 convolutional layers and 2 fully-connected layers. It divides the input image into a $s \times s$ grid with k bounding boxes per grid. It assigns a *confidence score* for each bounding box: $\mathcal{C} = \texttt{P}(obj) \cdot \texttt{IOU}(true, pred)$. Here, $\texttt{P}(obj)$ is the probability of an object in the bounding box and IOU is the intersection over union of the true bounding box and the predicted bounding box. It also predicts a *class probability* for each bounding box. We train the YOLO model by: (1) minimizing the bounding box prediction error to ensure $\texttt{IOU}(true, pred) \rightarrow 1$, and (2) maximizing the probability of correct class predictions.

While YOLO is accurate on challenging datasets, it is computationally expensive and requires multiple GPUs for real-time operation (*e.g.*, 40 fps). Furthermore, the model is designed for dense, generalized object detection on the COCO dataset that contains a wide array of classes [21]. The resultant model complexity is not justified when it is geared towards a particular domain with a narrow set of classes (*e.g.*, dashboard camera videos in BDD). In this scenario, specialized models deliver higher performance.

YOLO-Specialized. To construct a specialized YOLO model that is specialized to one cluster, we first prune a subset of convolutional layers from the original model while preserving sufficient accuracy on the given task and dataset. The resulting model, which we refer to as YOLO-SPECIALIZED, is capable of object detection with fewer computational resources and a smaller memory footprint. SPECIALIZER trains the YOLO-SPECIALIZED model on the data points in a particular cluster. ODIN builds specialized models for each detected cluster. Since it optimizes these models for a subset of the data space, they are smaller and support faster inference. Unlike the baseline network, YOLO-SPECIALIZED only contains 9 convolutional layers. Since these models are smaller, they do not suffer from the vanishing gradient problem during training. Thus, we remove the batch normalization layer from the network.

YOLO-Lite. To construct a lite YOLO model, we use the YOLO-SPECIALIZED model architecture and train it with the outputs of the original YOLO model. We refer to this lite model as YOLO-LITE. This model approximates its teacher's accuracy (*i.e.*, YOLO) at higher throughput. Compared to YOLO-SPECIALIZED, YOLO-LITE is easier to train since it does not require externally sourced oracle labels. SPECIALIZER directly use the outputs of the YOLO model on the newly detected cluster to train a YOLO-LITE model.

5.3 Model Selection

After DETECTOR and SPECIALIZER have identified the new clusters and generated specialized models, ODIN relies on the SELECTOR to pick the appropriate specialized models for prediction. Typically, for a given point, SELECTOR chooses the specialized model associated with that point's cluster. However, in the presence of drift, DETECTOR is actively assigning points to a new cluster and

SPECIALIZER is yet to construct a model for that cluster. In this scenario, SELECTOR must choose amongst the existing specialized models associated with closely-related clusters.

SELECTOR employs a model ensemble selection policy: $S_k : x_i \rightarrow \{M\}^k$, to pick k best-fit models to operate on x_i. We examine the following selection policies:

- **k-nearest models: unweighted (KNN-U)**. Under this policy, SELECTOR picks the k nearest models based on distance between x_i and the cluster centroids.
- **k-nearest models: weighted (KNN-W)**. Under this policy, SELECTOR picks the same set of models as the prior policy. However, it prioritizes these models based on the distances $\{d\}^k$ between their clusters' centroids and x_i. The weights are inversely proportional to distance (*i.e.*, the closest cluster gets the highest priority):

$$w_m = d'_i / \sum d'_i \tag{8}$$

  Here, w_m is the weight of the model associated with cluster m and $d'_i = \texttt{max}(\{d\}^k)/d_i$ is the inverted distance.
- **Δ-band models (Δ-BM)**. Under this policy, SELECTOR picks the models associated with all of the clusters whose Δ-bands contain x_i. If x_i does not fall within any Δ-band, then SELECTOR falls back to the KNN-W policy.

6. EVALUATION

Our evaluation of ODIN aims to answer the following questions:

- Is DETECTOR effective at identifying drift compared to the state-of-the-art outlier detection algorithms? (§6.2)
- Are the models constructed by SPECIALIZER effective and efficient in comparison to the baseline model? (§6.3)
- How do the model selection policies employed by SELECTOR cope with drift? (§6.4)
- How does ODIN perform in the presence of drift? (§6.5)
- How does ODIN execute end-to-end queries? (§6.6)
- What is the impact of each component of ODIN on its efficacy? (§6.7)

6.1 System Setup

Implementation. We implement ODIN in Python 3.6. We develop all of the convolutional neural networks using PyTorch 1.4 [28]. We leverage an off-the-shelf implementation of YOLOv3, and modify its layers to construct YOLO-Tiny. We use the MS-COCO API to operate on the BDD dataset [21].

Machine. We perform our experiments on a server with an NVIDIA Tesla P100 (16 GB RAM) and an Intel Xeon 2GHz CPU (2 threads). The server contains 12 GB of RAM.

Datasets. We use the following datasets to evaluate ODIN.

❶ **MNIST:** This dataset consists of 60K 28×28 black-and-white images of handwritten digits[4]. We use this dataset to highlight the properties of the latent space associated with standard and adversarial AEs in Figure 2a and Figure 2b. We validate the drift detection algorithm on this dataset in §6.2.

❷ **CIFAR-10:** This dataset consists of 60K 32×32 colored images belonging to ten classes[18]. We also use this dataset to validate the drift detection algorithm.

❸ **BDD:** This dataset consists of 100K 1280×720 colored images obtained from dashboard camera videos [38]. These high-resolution images are captured under diverse environments:

- **Time of day:** dawn, day, and night.
- **Weather:** rainy, snowy, foggy, cloudy, and overcast conditions.
- **Location:** residential, highway, city, and other locations.

There are ten classes of objects in BDD (*e.g.*, traffic lights, cars). We use this dataset to validate the efficacy of ODIN across drifting environmental conditions. We initially train models on a subset of the BDD dataset. We then introduce unseen clusters in BDD during our evaluation to examine ODIN's drift detection and recovery capabilities. When DETECTOR detects drift due to the novelty of unseen clusters, the SPECIALIZER starts generating new models for these subsets and the SELECTOR picks them once they are generated.

Dimensionality. Each dataset's dimensionality is the number of pixels in each image. The dimensionality of images in MNIST, CIFAR-10, and BDD is 784, 1024, ∼921K, respectively.

Table 1: Impact of Distance Metric on Drift Detection Accuracy: We compare the accuracy of DA-GAN (DG) in DETECTOR against approaches on MNIST and CIFAR-10: (1) LOF, (2) DRAE, (3) AE, (4) adversarial AE (AAE), and PCA. We reproduce the accuracy scores of LOF[2] and DRAE[36].

Outliers	MNIST						CIFAR-10		
	LOF	DRAE	AE	AAE	PCA	DG	AE	AAE	DG
0%	0.95	0.98	0.98	0.98	0.82	0.99	0.91	0.98	0.99
10%	0.92	0.95	0.93	0.97	0.69	0.98	0.84	0.97	0.97
20%	0.83	0.91	0.90	0.83	0.61	0.97	0.82	0.95	0.97
30%	0.72	0.88	0.87	0.91	0.31	0.96	0.81	0.93	0.95
40%	0.65	0.82	0.84	0.91	0.30	0.95	0.77	0.91	0.95
50%	0.55	0.73	0.82	0.90	0.29	0.94	0.74	0.89	0.94

6.2 Drift Detection

In this experiment, we measure the efficacy of DETECTOR (§4). We first compare the F1-score of the DA-GAN distance metric on two datasets against that delivered using other distance metrics. We compare DA-GAN against two state-of-the-art distance metrics that are geared towards low-dimensional data: (1) LOF [2], and (2) DRAE [36]. We also compare it against PCA, a canonical dimensionality reduction technique [40]. LOF estimates density of the input space and clusters regions of similar density. It detects drift by comparing the density distribution of recent points to that of the training data. DRAE uses the reconstruction error of an AE on the high-dimensional output images to detect drift.

Since DA-GAN models the distribution using a low-dimensional manifold for identifying drift, it is more robust to the curse of dimensionality. We configure $\Delta = 0.75$ in DA-GAN (Figure 4). We train DA-GAN for 100 epochs with a learning rate of 0.003 using the adversarial training procedure in Algorithm 1.

MNIST. We configure two digits to be outlier classes. We vary the percentage of outliers in the test dataset from 0% through 50%. The results are shown in Table 1. The most notable observation is that LOF and DRAE metrics do not scale up even to the comparatively low-dimensional images in MNIST. PCA exhibits lower accuracy since it does not take the spatial locality of the pixels in the image into consideration. As we increase the percentage of outliers to 50%, the accuracy of LOF and DRAE drops to 0.73 and 0.55, respectively, and PCA drops to 0.28%. In case of LOF, we attribute this to its reliance on the nearest neighbor distance. In case of DRAE, this is because it directly uses the reconstruction error on the output images. DA-GAN detects outliers more effectively by projecting the inputs to a low-dimensional manifold, since it captures the information in the image with the GAN. As we increase the percentage of outliers to 50%, accuracy of DA-GAN only drops from 0.99 to 0.94.

CIFAR-10. We conduct a similar empirical analysis on CIFAR-10. Prior work on outlier detection has only focused on cross-class accuracy differences in this dataset (and not on the percentage of

Table 2: Distribution of Images: We compute the distribution of the images in the BDD dataset across the four clusters identified by DETECTOR in an unsupervised manner based on their true labels. A subset of images in the dataset are not labeled (*undefined*).

Clusters	**Clear** (57428 imgs)			**Foggy** (143 imgs)			**Overcast** (10009 imgs)			**Rainy** (5795 imgs)			**Snowy** (6316 imgs)			**Undefined** (20309 imgs)
	Dawn	Day	Night	Dawn	Day	Night	Dawn	Day	Night	Dawn	Day	Night	Dawn	Day	Night	
C-α	90%	99%	0%	0%	21%	0%	58%	75%	0%	0%	10%	0%	15%	1%	0%	61%
C-β	0%	0%	100%	0%	0%	100%	0%	0%	100%	0%	6%	100%	0%	0%	100%	35%
C-γ	0%	0%	0%	63%	23%	0%	41%	18%	0%	100%	80%	0%	28%	0%	0%	3%
C-δ	9%	1%	0%	38%	57%	0%	1%	7%	0%	0%	3%	0%	57%	99%	0%	0%

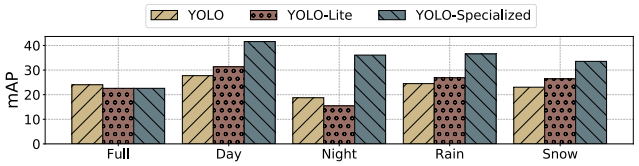


Figure 8: Impact of Model Specialization on Accuracy: We compare the detection accuracy (mAP metric) of the static YOLO model against the models constructed by SPECIALIZER: YOLO-LITE and YOLO-SPECIALIZED.

Table 3: Impact of Model Specialization on Cross-Subset Detection Accuracy: We compare the cross-subset detection accuracy of the YOLO model against the models constructed by SPECIALIZER: YOLO-SPECIALIZED and YOLO-LITE.

Data	**Cluster used for Specialization**				
	Baseline	C-α	C-β	C-γ	C-δ
FULL-DATA	0.2403	0.2068	0.2215	**0.2581**	0.2445
DAY-DATA	0.2772	**0.4157**	0.2229	0.2900	0.3339
NIGHT-DATA	0.1875	0.0789	**0.3609**	0.2691	0.2439
RAIN-DATA	0.2449	0.2424	0.2645	**0.3656**	0.3223
SNOW-DATA	0.2304	0.2082	0.2467	0.2636	**0.3354**

outliers). So, we compare DA-GAN against AE and adversarial AE (AAE) distance metrics. These metrics outperform LOF and DRAE metrics on MNIST. For instance, on MNIST, when the percentage of outliers is 50%, the accuracy of AE and AAE metrics only drop to 0.82 and 0.90, respectively. On CIFAR-10, the accuracy of AE and AAE metrics drop to 0.74 and 0.89, respectively. We attribute this to the higher dimensionality of images in CIFAR-10 compared to that in MNIST. The adversarial AE metric outperforms its standard counterpart by circumventing the irregular mapping problem (§2.3). DA-GAN outperforms AE and AAE metrics on this dataset. As we increase the percentage of outliers to 50%, its accuracy only drops from 0.99 to 0.94.

BDD. We next evaluate the efficacy of DETECTOR on the BDD dataset. This is a challenging dataset with a high-dimensional manifold. We only use the DA-GAN metric in this experiment. We train the DA-GAN on a held-out subset of BDD consisting of ∼20 K images. These images do not have any time of day or weather labels associated with them (*i.e.*, `undefined` images in Table 2).

We seek to examine its ability to detect images from previously unseen classes. The dataset contains 15 labeled subsets based on different environmental conditions. We note that the time of day and weather attributes of a video are independent (*e.g.*, video collected on a snowy night)[2]. We construct a workload that exhibits gradual drift by introducing images from the outlier subsets.

DETECTOR identifies drift using unsupervised clustering with the DA-GAN distance metric (*i.e.*, without using the time of day and weather attributes of images). It automatically learns four clusters out of the 15 subsets. The results are summarized in Table 2. We compute the distribution of the images across the detected clusters based on their labels to examine why DETECTOR picked these clusters. C-α mostly contains images captured on clear days as well as a few overcast images that are tagged as partially cloudy. DETECTOR groups nearly all of night-time images into C-β. C-γ mostly contains images with rain (as well as some images with snowfall and fog). C-δ mostly contains images with snowfall along with a few images with fog (*e.g.*, *fog-day*, *fog-night* pairings). The distribution of images across these clusters indicate that DETECTOR identifies the key features of the dataset. Among the 15 labeled subsets of the BDD dataset, DETECTOR automatically subsumes similar subsets into the same cluster. For instance, it maps nearly all of the night-time images (∼ 98%) to C-β, irrespective of the weather condition.

BDD Clusters. Using the clusters obtained in this experiment, we construct five data subsets that we leverage for testing in later experiments: (1) all of the images (FULL-DATA, 79863 images [3]), (2) images captured during day-time under clear weather conditions (DAY-DATA, 40696 images), (3) images captured during night-time under any weather condition (NIGHT-DATA, 31900 images), (4) images captured under rainy or overcast weather conditions (RAIN-DATA, 5808 images), and (5) images captured under snowy weather conditions (SNOW-DATA, 6313 images).

6.3 Model Specialization

In this experiment, we examine the efficacy of the models constructed by the SPECIALIZER in ODIN for each of the four detected clusters.

Specialized vs. Lite models. We first examine the detection accuracy of the three object detector models (§5.2): YOLO, YOLO-SPECIALIZED, and YOLO-LITE. We train and test the models over the five BDD clusters.

Detection Accuracy. The results are shown in Figure 8. The most notable observation is that YOLO-SPECIALIZED is the best performing model across all subsets (except for FULL-DATA). For each cluster, YOLO-SPECIALIZED delivers higher detection accuracy that its counterparts since it is specialized only on that subset. For example, on NIGHT-DATA, the YOLO-SPECIALIZED model delivers 2× higher accuracy compared to YOLO. It improves accuracy by 1.5× on average across all clusters.

SPECIALIZER directly trains the YOLO-LITE student model using the outputs of YOLO without requiring externally sourced labels. So its detection accuracy is comparable to YOLO across most subsets. YOLO-LITE's detection accuracy on NIGHT-DATA than that of YOLO. This is because YOLO makes most of its mistakes on this challenging subset. Since YOLO-LITE is smaller than YOLO, it does not learn all of the features of NIGHT-DATA.

[2]DETECTOR found that the location attribute is not important from a drift detection standpoint.

[3]BDD contains three splits of 69863, 20137, and 10000 images each. We set aside the second split to train non-specialized models.

Table 4: Impact of Model Specialization on Performance and Memory Footprint: We compare the performance and memory footprint of the baseline YOLO model against the models constructed by SPECIALIZER: YOLO-SPECIALIZED and YOLO-LITE.

Model	Architecture[30]	Throughput	Size
YOLO	YOLOv3	24 FPS	237MB
YOLO-SPECIALIZED	Pruned YOLOv3-tiny	144 FPS	34MB
YOLO-LITE	YOLOv3-tiny	140 FPS	35MB

Cross-Subset Detection Accuracy. We next examine the detection accuracy (*i.e.*, mAP score) of the specialized YOLO-SPECIALIZED models on other subsets that they are not trained on. Due to class imbalance in the BDD dataset, we train each model on the same number of samples (constrained by the smallest cluster). As shown in Table 3, each specialized model outperforms the YOLO model on their target subset. For instance, consider the YOLO-SPECIALIZED model trained on C-α. On DAY-DATA, it delivers 2× higher detection accuracy than the model tailored for C-β. It also works well on RAIN-DATA and SNOW-DATA since most of the data in these subsets are taken during the day. However, it delivers 5× lower detection accuracy on the NIGHT-DATA in comparison to C-β. This is because most of the training data for C-α are captured on clear days, which is different from NIGHT-DATA.

Model Generation Time. Since ODIN automatically clusters the dataset, each cluster contains more homogenous data points compared to the entire dataset. So, it is able to quickly generate smaller YOLO-LITE and YOLO-SPECIALIZED models on these clusters. For example, on NIGHT-DATA, ODIN generates a YOLO-SPECIALIZED model 21× faster compared to an off-the-shelf unspecialized YOLO model. The reasons for this are twofold. First, the specialized model consists of 7× fewer parameters compared to the original YOLO model. Second, the NIGHT-DATA cluster contains 3× fewer images compared to FULL-DATA. Thus, reduction in model and dataset sizes enable faster training of specialized models.

Query Execution Time. As shown in Table 4, ODIN delivers higher throughput by leveraging these models. Since YOLO-SPECIALIZED and YOLO-LITE are 7× smaller than YOLO, they are nearly 6× faster than the YOLO model.

Memory Footprint. Since SPECIALIZER constructs four models on the BDD dataset, the overall memory footprint of ODIN is 2× smaller than with the baseline YOLO model.

When DETECTOR finds a new cluster, SPECIALIZER first generates a YOLO-LITE model using the outputs of YOLO. YOLO-LITE's advantage over YOLO-SPECIALIZED lies in faster training as there is no need to wait for externally sourced labels. When the labels are available, either from humans or using weak-supervision [29], SPECIALIZER constructs a YOLO-SPECIALIZED model that delivers higher detection accuracy than its YOLO-LITE counterpart. In the rest of the experiments, we configure ODIN to use YOLO-SPECIALIZED models.

6.4 Model Selection

In this experiment, we compare the efficacy of the model selection policies discussed in §5.3 on the BDD dataset:

❶ **KNN-U:** SELECTOR uses the unweighted average of the four models constructed by SPECIALIZER.

❷ **KNN-W:** SELECTOR uses the weighted average of the four models constructed by SPECIALIZER. SELECTOR computes weights by normalizing the distances in the latent space obtained using DA-GAN using Equation 8.

❸ **Δ-BM:** With this policy, SELECTOR uses the high-density Δ-bands of cluster while picking models. For an image that falls inside a Δ-band, we select the model associated with that Δ-band's cluste. For an image that falls outside any of existing Δ-bands, we revert to the KNN-W policy (8% of the images in BDD). For an image the falls inside multiple overlapping Δ-bands, we use all of the bands with equal weights (39% of the images in BDD).

Table 5: Impact of Model Selection on Accuracy: We compare the policies adopted by SELECTOR for picking the specialized YOLO-SPECIALIZED models compared to the baseline YOLO model.

Data	Model Selection Policy			
	Baseline	KNN-U	KNN-W	Δ-BM
FULL-DATA	0.2403	0.2365	**0.2811**	0.2491
DAY-DATA	0.2772	0.3514	0.3954	**0.4257**
NIGHT-DATA	0.1875	0.2123	0.3432	**0.3687**
RAIN-DATA	0.2449	0.2843	**0.3764**	0.3552
SNOW-DATA	0.2304	0.3134	0.3412	**0.3653**

The results are shown in Table 5. Our baseline is the a static system without drift detection or recovery. KNN-W outperforms KNN-U on all of the subsets. For instance, on RAIN-DATA, it delivers 32% higher detection accuracy compared to KNN-U. This is because it ensures that the best-fit model (*i.e.*, the one specialized on C-γ) is given the highest consideration.

Δ-BM policy outperforms KNN-W on most of the subsets. For instance, on DAY-DATA, it delivers 7.5% higher detection accuracy compared to KNN-W. We attribute this to Δ-BM policy's focus on high-density bands instead of the entire clusters (as KNN-U and KNN-W do). This policy works well in tandem with DETECTOR that leverages Δ-bands to identify drift. On RAIN-DATA, KNN-W outperforms Δ-BM by 6%. This is because Δ-BM only uses the high-density bands of C-γ, since it contains most images with rain. However, C-γ does not contain images with cloudy skies. So the model trained on this cluster is slightly less effective on RAIN-DATA. KNN-W circumvents this limitation by using all of the modelss.

6.5 End-to-End Evaluation

We next examine the efficacy and efficiency of all of the components of ODIN in tandem. We evaluate ODIN under three configurations on a sequence of 100 K images in BDD. We construct the sequence thus: (1) 20 K images exclusively from NIGHT-DATA images, (2) after 20 K images, we add DAY-DATA to the pool. (3) after 40 K images, we add SNOW-DATA to the pool, and (4) after 60 K images, we add RAIN-DATA to the pool. The chance for selecting an image of any subset is not adjusted for equal chance, since we want to replicate a realistic distribution. We measure the object detection accuracy (mAP) of ODIN every 5 K images in the sequence. The results are shown in Figure 9.

❶ **Baseline:** In the baseline configuration, ODIN uses a single YOLO model to process the entire sequence of images. Without drift recovery, this system's detection accuracy is ∼20 mAP. This is because it is unable to detect and recover from drift. ODIN processes images at 24 FPS under this configuration. Since there are no specialized lightweight models, its performance is constrained by the throughput of the heavyweight YOLO model (see Table 4)

❷ **Δ-BM:** We next enable drift recovery and configure ODIN to use the Δ-BM selection policy. In this configuration, ODIN first uses a YOLO-LITE model to process the NIGHT-DATA images. The accuracy is comparable to the baseline. This is because YOLO-LITE delivers similar accuracy to the full model (Figure 8). When DETECTOR identifies a new cluster, ODIN generates a YOLO-SPECIALIZED model and switches to it (as it outperforms the YOLO-LITE and full models). Each of the dotted lines in Figure 9 represents identification of a new cluster by DETECTOR and subsequent generation of a YOLO-SPECIALIZED model. The spe-

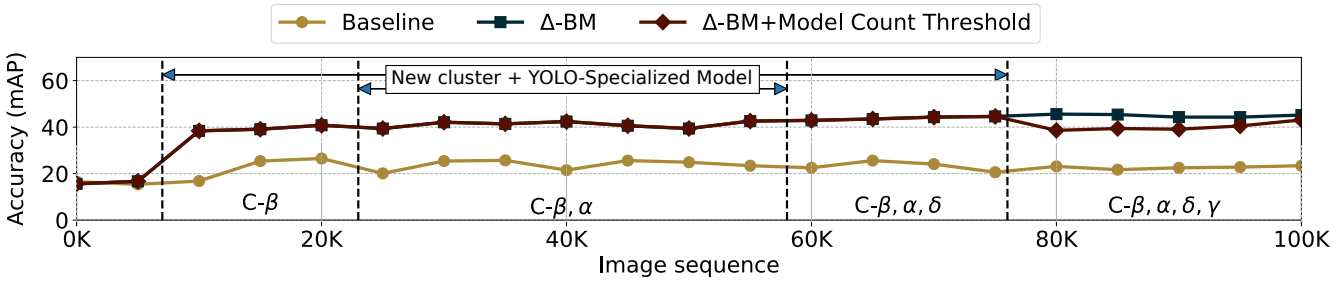


Figure 9: End-to-End Evaluation: We examine detection accuracy of ODIN with all components under three configurations. ❶ **Baseline** A large YOLO model is used to process all BDD videos. ❷ **Δ-BM:** We enable drift recovery and configure ODIN to use the Δ-BM selection policy. ❸ **Δ-BM + Model Count Threshold:** We next limit the maximum number of models to three.

Table 6: Aggregation queries and Lightweight Filters: We compare the efficacy and efficiency of executing aggregation queries across several configurations. These configurations include: (1) a static system without specialized models, (2) ODIN-HEAVY that uses specialized YOLO models, (3) ODIN with no filters and YOLO-SPECIALIZED models, (4) ODIN-PP with unspecialized filters and YOLO-SPECIALIZED models, and (5) ODIN-FILTER with specialized filters and YOLO-SPECIALIZED models.

	Architecture	Metric	Cars	Trucks	FPS
Aggregation Queries	Static	Query Acc.	0.65	0.86	24
	ODIN	Query Acc.	0.94	0.92	140
	ODIN-HEAVY	Query Acc.	0.97	0.98	20
Aggregation Queries with Filters	ODIN-FILTER	Query Acc. Reduction	0.92 8%	0.83 68%	130
	ODIN-PP	Query Acc. Reduction	0.59 38%	0.76 76%	135

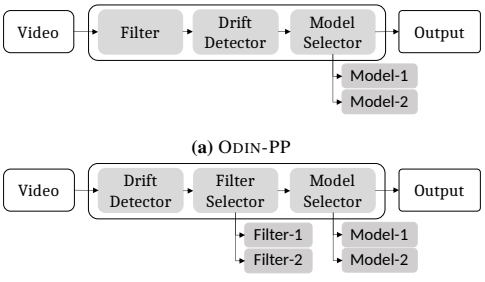


Figure 10: ODIN Configurations: We augment ODIN with lightweight DNN filters to improve throughput [23]. (1) ODIN-PP with specialized models and unspecialized filter. (2) ODIN-FILTER with specialized models and specialized filters.

cialized models double the detection accuracy from ∼20 mAP to ∼40 mAP. This is because the SELECTOR picks the appropriate model constructed by SPECIALIZER using the Δ-BM policy.

❸ **Δ-BM + Model Count Threshold:** We next limit to the maximum number of models to three. When DETECTOR identifies the fourth cluster (*i.e.*, C-γ), it drops the cluster with the smallest number of inputs. In this dataset, it drops C-δ (5 K images in cluster). Since it only relies on the three other models for prediction, its detection accuracy suffers slightly due to the missing model. With the Δ-BM policy, SELECTOR reverts to KNN-W when encountering points outside the existing Δ bands. So, the drop in detection accuracy is not significant. The throughput is slightly higher due to fewer models (4 → 3), at 140 FPS

6.6 Aggregation Query

We next examine how ODIN complements the filtering technique used in state-of-the-art visual DBMSs [15, 23]. We focus on aggregation queries (*e.g.*, number of cars in a set of videos):

```
SELECT FROM (SELECT detections
        FROM bdd USING MODEL yolo_specialized
        WHERE class='car')
```

We consider two classes: cars and trucks. In each case, SELECTOR selects the appropriate YOLO-SPECIALIZED model for each image. We compare ODIN against two systems: (1) a static system without specialized models, and (2) a variant of ODIN that uses specialized YOLO models instead of specialized YOLO-SPECIALIZED models. We refer to the latter variant as ODIN-HEAVY. The specialized YOLO models used by ODIN-HEAVY are 6× larger and slower than YOLO-SPECIALIZED models.

As shown in Table 6, ODIN returns more accurate results compared to the static system. For cars, ODIN and ODIN-HEAVY are 50% better than a static system. For trucks, which are larger objects, all systems perform better. While ODIN-HEAVY is slightly more accurate than ODIN (∼3-6%), it is 7× slower.

Aggregation Queries Using Lightweight Filters. We next examine how to accelerate the aggregation queries using lightweight filters [23]. In this case, the system returns approximate aggregates. ODIN creates specialized filters for each cluster. We modify the architecture of ODIN to incorporate these filters, as shown in Figure 10. The filter is a lightweight DNN that preprocesses the images to return a boolean decision that indicates whether that image must be subsequently processed by the heavyweight model (*e.g.*, YOLO-SPECIALIZED). In our example, a DNN with 3 convolutional layers is sufficient to determine if a given frame contains a car or not. If the frame has no cars, then ODIN-FILTER does not process it with the YOLO-SPECIALIZED model that counts the number of cars. In this case, the query looks thus:

```
SELECT COUNT(detections)
   FROM (SELECT detections
         FROM (SELECT * FROM bdd
               USING FILTER car_filter
               WHERE class=1))
         USING MODEL yolo_specialized
         WHERE class='car'
```

We consider three configurations: (1) ODIN with specialized models and no filters, (2) ODIN-PP with specialized models and unspecialized filter [23], and (3) ODIN-FILTER with specialized models and specialized filters. The results are shown in Table 6. With ODIN-FILTER, there is 8% data reduction for 'cars' (since cars are present in nearly every frame). Query accuracy slightly drops since the filter returns some false negatives. With trucks, we observer higher data reduction since they are rarer in BDD. The drop in query accuracy is more prominent with ODIN-PP since it uses a single unspecialized filter. In the presence of drift, this filter returns

Table 7: Ablation study for ODIN: We delineate the impact of each component of ODIN.

Experiment	mAP	Query Acc	Throughput	Memory
End-to-End Model	40.15	93.5	140FPS	148MB
-SELECTOR	24.84	71.4	140FPS	148MB
Baseline	24.03	64.6	24FPS	237MB

more false negatives. With trucks, the filters miss more frames in NIGHT-DATA due to lighting conditions. This experiment shows that drift detection and recovery is important for filters as well.

6.7 Ablation Study

We next conduct an ablation study to delineate the impact of each component of ODIN. Since the DETECTOR is not useful without the recovery components, we consider these configurations:

- End-to-End System: With all three components.
- - SELECTOR: With only the DETECTOR and SPECIALIZER components. ODIN uses the most recently created YOLO-SPECIALIZED model in this setting.
- Baseline: Lastly, we remove all the three components. In this configuration, ODIN uses the heavyweight YOLO model.

We summarize the results in Table 7. Eliminating the SELECTOR leads to a drop in accuracy, since the best model is no longer used for each cluster. The naive model selection policy is only useful when the drift is monotonically increasing. In practice, older clusters co-exist with newer clusters, as is the case in BDD. Since the most recent model is trained on newer clusters, its accuracy drops when older clusters are re-introduced. The memory footprint and throughput are nearly unchanged, since the SELECTOR is computationally lightweight. Lastly, removing the DETECTOR and SPECIALIZER is equivalent to using a static heavyweight YOLO model. The lack of specialization leads to lower accuracy. Furthermore, performance also suffers since the YOLO model is larger and slower than the YOLO-SPECIALIZED models constructed by the SPECIALIZER.

7. LIMITATIONS

We now discuss the limitations of ODIN and present our ideas for addressing them in the future.

Availability of Oracle Labels. In ODIN, we assume that oracle labels are available for images in newly detected clusters. In practice, these labels may not be available quickly if they are collected from humans. ODIN could circumvent this problem by first constructing fast YOLO-LITE models using the outputs of the pre-trained YOLO model, thereby bypassing the label availability constraint. While these models deliver performance comparable to their YOLO-SPECIALIZED counterparts, they suffer from lower accuracy on newly detected clusters. After the labels are obtained, ODIN trains YOLO-SPECIALIZED models and replaces their YOLO-LITE counterparts with these newly trained models. Weak supervision techniques may accelerate the procurement of oracle labels [29].

DA-GAN Performance. The performance of DA-GAN drops over time as the number of clusters increase. This is because it needs to compare each input against all of the Δ-bands associated with these clusters. For instance, in Figure 9, we observe a 5 FPS drop with four clusters. We believe that locality-sensitive hashing [7] might alleviate this problem. Another alternative is to design a more efficient model architecture for the encoder in DA-GAN, thereby reducing the time taken to encode a point.

8. RELATED WORK

Drift Detection. [6] presents a survey of several supervised drift detection mechanisms Unsupervised methods that detect drift based on the expected data distribution include model confidence methods [13, 33] and clustering algorithms [35, 22]. Outlier detection algorithms detect drift in low-dimensional structured data (Figure 5). DRAE [36] uses the reconstruction error of an AE to detect drift. Since AEs suffer from holes in their latent space, DRAE is only effective for static low-dimensional datasets. LOF [2] measures the density of the input space and clusters regions of similar density. It detects drift by comparing the density distribution of recent points to that of the training data. Researchers have also proposed windowing algorithms to adapt models when the type of drift is not known [22]. These algorithms use static windows to track changes in distribution. Unlike these techniques, ODIN generalizes to unstructured data. The reasons for this are threefold. First, DA-GAN represents high-dimensional data better than the AE in [36]. Second, Δ-bands compare high-density regions better than kNN in [2]. Lastly, it dynamically generates clusters over time instead of using static windows employed in [22].

Model Specialization. Recovering from drift is key to maintaining the accuracy of the overall system. ODIN relies on model specialization for drift recovery. It deploys models specialized for each detected cluster of the data space. Model distillation is a widely-used technique for specialization [27, 37]. With distillation, a teacher model trains a lite (*i.e.*, smaller and faster) student model to mimic its output. It is useful in scenarios where the teacher model is unlikely to fail (*i.e.* no drift). Model compression is another technique for specialization [3]. With compression, we start with a pre-trained model and prune weights below a threshold to reduce size. A pre-trained model is not effective in the presence of drift. Different from these techniques, ODIN relies on specialized models for specialization, where the models are trained from scratch on the novel data points. This enables it to work well on drifting datasets.

Model Selection. Given a collection of specialized models, it is important to choose the appropriate ones for processing an input. Prior efforts on model selection are geared towards low-dimensional data. ARF constructs an ensemble of weak decision trees and dynamically prune trees whose accuracy degrades due to drift [8]. It uses a simple majority technique to weight the ensemble of trees. KME combines several drift detectors to identify cyclical, real, and gradual drift occurrences [31]. It updates the models if it detects drift or if enough training data is collected for an update, and assigns weights using a model-to-concept mapping. It assigns higher weights to models that have been identified to deliver higher accuracy on recent concepts. These methods do not work well on high-dimensional data. ODIN uses SELECTOR, which uses either the Δ-DM policy for picking an ensemble of specialized models for processing a given input.

9. CONCLUSION

In this paper, we presented the architecture of ODIN, a system for detecting and recovering from drift in visual data analytics. We presented an unsupervised algorithm for drift detection by determining the high-density regions of the input data space. We proposed the DA-GAN distance metric that allows the DETECTOR to work well on high-dimensional data. ODIN constructs smaller, faster specialized models for each detected cluster that deliver higher accuracy compared to the larger, slower model trained on the entire dataset. Our evaluation shows that ODIN delivers higher throughput, higher detection and query accuracy, as well as a smaller memory footprint compared to the static setting without drift detection and recovery.

10. REFERENCES

[1] P. R. Almeida, L. S. Oliveira, A. S. Britto Jr, and R. Sabourin. Adapting dynamic classifier selection for concept drift. *Expert Systems with Applications*, 104:67–85, 2018.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *SIGMOD*, volume 29, pages 93–104. ACM, 2000.

[3] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *ACM SIGKDD*, pages 535–541. ACM, 2006.

[4] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[5] T. Falk, D. Mai, R. Bensch, Ö. Çiçek, A. Abdulkadir, Y. Marrakchi, A. Böhm, J. Deubner, Z. Jäckel, K. Seiwald, et al. U-net: deep learning for cell counting, detection, and morphometry. *Nature Methods*, 16(1):67–70, 2019.

[6] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44, 2014.

[7] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.

[8] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, 2017.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[11] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *IEEE/ACM Symposium on Edge Computing*, pages 115–131. IEEE, 2018.

[12] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD*, pages 11–22, 2004.

[13] H. Jiang, B. Kim, M. Guan, and M. Gupta. To trust or not to trust a classifier. In *NeurIPS*, pages 5541–5552, 2018.

[14] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. Chameleon: scalable adaptation of video analytics. In *SIGCOMM*, pages 253–266. ACM, 2018.

[15] D. Kang, P. Bailis, and M. Zaharia. Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics. *PVLDB*, 13(4):533–546, 2019.

[16] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: Optimizing neural network queries over video at scale. *PVLDB*, 10(11):1586–1597, 2017.

[17] S. Krishnan, A. Dziedzic, and A. J. Elmore. Deeplens: Towards a visual data management system. *CIDR*, 2019.

[18] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[20] Q. Lao, X. Jiang, M. Havaei, and Y. Bengio. Continuous domain adaptation with variational domain-agnostic feature replay. *arXiv preprint arXiv:2003.04382*, 2020.

[21] T. Lin and P. Dollar. Ms coco api, 2016.

[22] V. Losing, B. Hammer, and H. Wersing. Knn classifier with self adjusting memory for heterogeneous concept drift. In *ICDM*, pages 291–300. IEEE, 2016.

[23] Y. Lu, A. Chowdhery, S. Kandula, and S. Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1493–1508, 2018.

[24] A. Makhzani and B. J. Frey. Pixelgan autoencoders. In *NeurIPS*, pages 1975–1985, 2017.

[25] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv*, 2015.

[26] M. Manana, C. Tu, and P. A. Owolawi. A survey on vehicle detection based on convolution neural networks. In *ICCC*, pages 1751–1755. IEEE, 2017.

[27] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian. Online model distillation for efficient video inference. *ICCV*, pages 3573–3582, 2019.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.

[29] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):1–22, 2019.

[30] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[31] S. Ren, B. Liao, W. Zhu, and K. Li. Knowledge-maximized ensemble algorithm for different types of concept drift. *Information Sciences*, 430:261–281, 2018.

[32] M. Roveri. Learning discrete-time markov chains under concept drift. *IEEE Trans. on Neural Networks and Learning Systems*, 2019.

[33] T. S. Sethi and M. Kantardzic. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99, 2017.

[34] T. S. Sethi and M. Kantardzic. Handling adversarial concept drift in streaming data. *Expert Systems with Applications*, 97:18–40, 2018.

[35] E. J. Spinosa, A. P. de Leon F de Carvalho, and J. Gama. Olindda: A cluster-based approach for detecting novelty and concept drift in data streams. In *ACM SIGAPP SAC*, pages 448–452. ACM, 2007.

[36] Y. Xia, X. Cao, F. Wen, G. Hua, and J. Sun. Learning discriminative reconstructions for unsupervised outlier removal. In *ICCV*, pages 1511–1519, 2015.

[37] S. You, C. Xu, C. Xu, and D. Tao. Learning from multiple teacher networks. In *SIGKDD*, pages 1285–1294, 2017.

[38] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv*, 2018.

[39] Z. Zhang, Y. Song, and H. Qi. Age progression/regression by conditional adversarial autoencoder. In *CVPR*, pages 5810–5818, 2017.

[40] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.