Efficient and scalable covariate drift detection in machine learning systems with serverless computing

Jaime Céspedes Sisniega[a], Vicente Rodríguez[b], Germán Moltó[b], Álvaro López García[a,*]

[a]*Instituto de Física de Cantabria (IFCA), CSIC-UC, Avda. los Castros s/n, Santander, 39005, Spain*
[b]*Instituto de Instrumentación para Imagen Molecular (I3M). Centro Mixto CSIC - Universitat Politècnica de València, Camino de Vera s/n, Valencia, 46022, Spain*

Abstract

As machine learning models are increasingly deployed in production, robust monitoring and detection of concept and covariate drift become critical. This paper addresses the gap in the widespread adoption of drift detection techniques by proposing a serverless-based approach for batch covariate drift detection in ML systems. Leveraging the open-source OSCAR framework and the open-source Frouros drift detection library, we develop a set of services that enable parallel execution of two key components: the ML inference pipeline and the batch covariate drift detection pipeline. To this end, our proposal takes advantage of the elasticity and efficiency of serverless computing for ML pipelines, including scalability, cost-effectiveness, and seamless integration with existing infrastructure. We evaluate this approach through an edge ML use case, showcasing its operation on a simulated batch covariate drift scenario. Our research highlights the importance of integrating drift detection as a fundamental requirement in developing robust and trustworthy AI systems and encourages the adoption of these techniques in ML deployment pipelines. In this way, organizations can proactively identify and mitigate the adverse effects of covariate drift while capitalizing on the benefits offered by serverless computing.

Keywords: Serverless computing, machine learning, drift detection, covariate drift, covariate shift

1. Introduction

Nowadays, deploying a machine learning (ML) model in production is a straightforward task, thanks to the existence of a large number of tools aimed to make the process easier. Frameworks such as KServe [1], Kubeflow [2], MLflow [3], BentoML [4], or OSCAR [5] have been released, making it possible to deploy, operate, scale, and monitor ML models. In parallel, a field called MLOps (ML Operations) [6] has emerged to transfer some of the best practices from the DevOps (Development Operations) field to ML. Apart from the well-known Continuous Integration/Continuous Delivery (CI/CD) processes, some of these best practices include monitoring models at inference time to detect as soon as possible if model performance is decaying or if the data being used differs significantly from the data used at training time. Generally speaking, this process is known as drift detection [7] and can be defined as the process of detecting if there is a significant change in the concept learned from an ML model (*concept drift*), or if there is a relevant change related to the feature distributions (*covariate shift*) concerning a reference dataset, which in real-world problems sometimes is the dataset used to train the model of study.

A common mistake when deploying and using an ML model in production is to make the assumption (either consciously or not) that the data that was used to train the model and the data being used to feed the model at inference time belong to the same distribution (i.e., data is stationary). However, in real-world ML applications, this is not always the case: rapidly changing environments (like in the Internet of Things area), incomplete training data, poorly calibrated sensors over time (providing inaccurate measurements), or a methodology change in data collection may happen, to cite some issues that can make data non-stationary. In these cases, and many more, drift detectors allow monitoring either the model performance or the feature distributions so that ML model users can detect deviations leading to a model performance decay or even the model malfunctioning. If such a situation is detected, an ML model operator can decide to decommission that model, to replace it with a more accurate one, or to warn about the model potential performance decay, aspects that are fundamental to ensure human oversight and technical robustness and safety of artificial intelligence and ML systems, a key requirement of Trustworthy Artificial Intelligence (AI) systems [8] as exposed by the High-Level Expert Group on Artificial Intelligence set up by the European Commission. Model monitoring through drift detectors should be one of the following requirements when building ML systems.

However, despite the obvious benefits of providing drift detection as a key part of any ML system, these techniques are not yet widespread among researchers, machine learning engineers, and data scientists, not to mention the final users. This work aims to fill this gap by proposing a serverless-based approach to detect covariate drift in production ML systems that process data in batch mode. To achieve this, we will exploit the

*Corresponding author: aloga@ifca.unican.es

OSCAR framework in combination with the Frouros drift detection library [9], building a comprehensive service for batch covariate drift detection. This approach enables more robust predictions without incurring penalties at inference time. The main contributions of our work are as follows:

- We review the background in drift detection for ML production systems, with a special focus on serverless computing.
- We propose an approach following the serverless computing model for ML systems, incorporating covariate drift detection methods that process data in batch mode to provide more robust predictions and results.
- We evaluate the proposed approach by employing an edge ML use-case where covariate drift is intentionally induced on three different datasets.

The remainder of this article is structured as follows. Section 2 introduces the background and related work in serverless computing for machine learning and drift detection areas. Section 3 describes our proposed approach, which we further evaluate in Section 4 through the study of a use case. Finally, conclusions and future work are laid out in Section 5.

2. Background and Related Work

2.1. Drift Detection in ML Systems

Once an ML model has been trained for a specific problem (either regression or classification), it can be deployed in a production system to provide inference results with real-world data, which the model has not seen before. However, when using an ML model to deliver predictions over unseen data, both the user and developer should consider several critical questions that may directly impact the model's performance: Will new data come from the same distribution (i.e., is the data stationary)? If not, how can we detect such situations to prevent performance degradation? Will the *concept* learned by the model change over time?

Providing answers to these questions leads us directly to the field of drift detection. In the scientific literature, various definitions of different types of drift exist, as noted by Moreno et al. in [10]. In our work, we will adhere to the **real concept drift** terminology given by Gama et al. in [11] for concept drift, in combination with the one used by Shimodaira in [12] for detecting **covariate shift**[1] using only the feature or covariate distributions.

Concept drift is often associated with detecting changes over an infinite data stream, where individual data samples are processed as soon as they arrive, in a streaming or online manner [13, 14]. However, offline or batch methods for detecting concept drift have also been well-documented in the literature [15, 16]. Similarly, covariate drift detection methods can operate both in a streaming/online mode [17] and through two-sample tests in a batch/offline approach [18].

As briefly stated in the introduction of this work (cf. Section 1), drift detection in ML systems can be defined as a method to detect when either of the following two situations, which can harm the performance of a model, materializes:

- There is a significant change in the *concept* previously learned by an ML model, known as *concept drift*.
- There is a significant change related to the feature or covariate distributions used to fit the ML model, known as *covariate drift*.

The existence of these deviations (in the concept or covariates) may result in a performance degradation of the model, leading to a loss in its classification, prediction, or decision-making capabilities. This is especially important when building robust and secure ML systems. If a drift situation occurs unnoticed, the system's reliability may be compromised, delivering misleading or inaccurate results.

Detecting drift in an ML model is of paramount importance for any ML stakeholder (developer, operator, user) so they can decide on the appropriate actions (e.g., stopping the inference process, refitting the model to the new data, etc.). There is a considerable body of work applying drift detection techniques [19] across a variety of domains, such as predictive maintenance [20], the Internet of Things (IoT) [21], social big data [22], and fake online reviews [23].

In recent years, multiple libraries that implement drift detection methods have emerged, such as Alibi Detect [24], MOA [25], River [26], and Frouros[2] [9]. Among these, Frouros stands out as the open-source Python library that implements the highest number of different drift detection methods. Additionally, it is designed to be used with any ML framework, making it framework-agnostic.

2.2. Serverless Computing

Serverless computing has emerged over the last years as a paradigm for event-driven computing on services where the service provider manages the resource allocation. In particular, the Functions as a Service (FaaS) model, exemplified by services such as AWS Lambda, allows users to define functions that execute in response to certain events to perform event-driven processing on the massively scalable platform managed by AWS. This has become an appropriate computing and programming paradigm for scientific applications, as exemplified in the work by Pérez et al. [27], where AWS Lambda is employed to execute deep learning frameworks and video processing tools.

AWS Lambda provides highly-elastic capabilities, where up to 3000 parallel invocations of a function can run concurrently, with up to 10 GB of RAM for each, including the automated mount of distributed filesystems shared among the multiple invocations. This has unlocked the ability to perform serverless

[1] Henceforth, we adopt the term *covariate drift* in place of *covariate shift* to ensure consistency with the terminology of *concept drift*.

[2] Frouros - `https://frouros.readthedocs.io`

distributed ML training without the need to pre-provision infrastructure.

However, to mitigate the lock-in and support, to some extent, a similar computational paradigm on existing hardware, and on-premises serverless platforms have also appeared in the last years, such as Knative, OpenFaaS, or OpenWhisk. As an illustrative example, [28] leverages OpenWhisk to provide serverless endpoints for the inference of machine learning models via a REST API [29]. In this line, OSCAR[3] [5] is an open-source platform to support the event-driven serverless paradigm to execute applications along the computing continuum. This leverages the ability to run on elastic Kubernetes clusters, which provide seamless resource provisioning for containerized user workloads. OSCAR can run on low-power devices such as Raspberry Pis, on-premises, and public Clouds and it is integrated with the open-source SCAR[4] [27] framework to create data-driven workflows from the edge of the network until FaaS platforms such as AWS Lambda. OSCAR is the framework adopted in this work to support event-driven serverless computing both for scalable inference of AI models and execution of services for drift detection. An OSCAR service can be triggered asynchronously by uploading a file to an object-storage system, such as MinIO, which creates a Kubernetes job that is executed in the underlying cluster, which can optionally grow and shrink in terms of the number of nodes. It can also be triggered synchronously, allowing a certain number of containers to be up and running, thanks to Knative, to mitigate the *cold start* problem (i.e., increased latency for the first invocations, due to resource provisioning). An OSCAR service is defined by a Docker image, stored in an image container registry (e.g., Docker Hub or GitHub Container Registry); a shell script that will be executed in a container dynamically created from the Docker image; a set of computing requirements, in terms of CPUs, RAM, GPUs, etc., and, optionally, the input bucket in the object-storage system that will trigger the service and where the output data should be stored from the supported storage back-ends (MinIO, dCache, Onedata, etc.).

In [30], Naranjo et al. present a framework where the inference of machine learning models can be executed in a public cloud or on-premises clouds following a serverless approach based on the aforementioned tools, where computational resources are dynamically provisioned according to the inference workloads.

2.3. Related Work

Regarding existing systems or approaches for drift detection using serverless computing, Rausch et al. [31] propose a serverless platform focused on edge AI that can monitor concept drift over certain metrics through a concept they call *quality gates*. In the work of Muthusamy et al. [32], feature vectors (covariate distributions) are used to detect covariate drift, but they rely only on algorithms that group features to detect anomalous inputs, such as K-means, and not on common covariate drift methods like those found in the libraries mentioned in Section 2. This approach can be combined with what is proposed by the same authors in [33] to adapt it to use serverless computing. The work by Thinakaran et al. [34] introduces a covariate drift [5] monitor to trigger model retraining at required intervals to reduce the accuracy gap. For this, serverless functions that run on AWS Lambda are employed to develop a cost-efficient continuous learning framework. The work by Zhang et al. [35] describes a serverless cloud-fog platform for video analytics with incremental learning. They tackle the issue of drift detection through limited human feedback into the system.

Setting aside serverless computing and considering existing systems for drift detection, Wang et al. [36] propose a recommendation system for cloud services API selection that is concept drift aware. In this work, they implement a detector based on the Jensen–Shannon distance and present the predicted results and the detected drift together. In [37], the authors present an experimental approach to handle concept drift in predictive process monitoring, in which the forecasting model adapts to concept drift automatically. An edge MLOps framework that employs multiple pipelines to establish a complete lifecycle for Artificial Intelligence of Things (AIoT) is proposed in [38]. One notable feature within these pipelines is the inclusion of a mechanism to address model drift (which may be caused by concept or covariate drift) but relies solely on the model error without using a drift detector. The authors of [39] propose an adaptive ML system for data analytics that is composed of an ensemble of an ML model and a drift detector for Internet of Things (IoT) online data streams, named Optimized Adaptive and Sliding Windowing (OASWW), making it possible to process online IoT data streams by adapting to potential concept drift. Similarly, Matchmaker is a solution proposed in [40] to handle a mix of drifts (concept drift and covariate shift) simultaneously for large-scale ML production systems. This solution adapts to drift by having multiple models, each trained on a different batch of data, which allows redirecting the test sample to the most similar model at inference time.

However, to our knowledge, none of these approaches propose an agnostic covariate drift detection system (from the detector's point of view) that processes data in batch mode follows the serverless architectural pattern and can run on multiple cloud back-ends using state-of-the-art covariate drift detectors.

3. Proposed Approach

To address the batch processing needs for detecting covariate drift while leveraging the benefits of serverless computing, we propose deploying an ML inference service pipeline in parallel with a covariate drift detection service pipeline, as depicted in Figure 1. Although both systems are coupled (i.e., the drift detection and the inference services are related), they can be used independently to avoid introducing a penalty in the inference process.

Each service will feature an input object-storage system (e.g., a bucket in MinIO) that will trigger the invocation of the service upon a file upload. The output result of each service can

[3] OSCAR - `https://oscar.grycap.net`

[4] SCAR - `https://github.com/grycap/scar`

[5] They refer to *covariate drift* as *data drift*.

be uploaded into the input object-storage system of another service, effectively creating an event-driven pipeline. For simplicity, these object-storage systems are omitted in Figure 1 but are shown in detail in Figure 2, which describes the use case presented in Section 4.

These services will be run on a serverless platform to benefit from event-driven execution and the scalability of automated resource provisioning from the underlying computing platform. It is important to note that these services do not exchange information via direct API calls. Function chaining via direct API calls is considered an anti-pattern according to the *Serverless trilemma* [41], as it would incur double-billing when using a FaaS (Functions as a Service) serverless service from a public cloud provider.

The different services are described as follows.

3.1. ML Inference Service (MLIS)

This service comprises a pre-trained ML model. Hence, its responsibility is to receive a single sample $x \in \mathbb{R}^m$, where m is the number of features, and perform inference to obtain a prediction $\hat{y} \in \mathbb{R}$ in the case of a regression problem or $\hat{y} \in \mathbb{R}^c$, where c is the number of classes, in a classification scenario.

In the specific scenario outlined in this manuscript, as we assess the approach using computer vision datasets detailed in Section 4.1.1, the MLIS receives image uploads through an object storage system to initiate the inference process. The service is triggered for each uploaded image. The rate of image upload determines the frequency of service invocation. Subsequently, the inference output is fed as input into the drift detection pipeline, with the Dimensionality Reduction Service (DRS) serving as its initial stage.

3.2. Dimensionality Reduction Service (DRS)

To address the curse of dimensionality problem [42], we introduce a service that takes an input sample $x \in \mathbb{R}^m$ and transforms it to $\hat{x} \in \mathbb{R}^k$, where $k \ll m$.

To accomplish this task, commonly used methods such as Uniform Manifold Approximation and Projection (UMAP) [43], Random Projection [44], Principal Component Analysis (PCA) [45], Kernel Principal Component Analysis (Kernel PCA) [46], or Autoencoders [47] can be employed.

Regardless of the chosen technique, a subset of samples is typically extracted from the data to fit the dimensional reduction method. This service receives the same image that triggers an inference process and performs the chosen dimension reduction technique. Therefore, the DRS is invoked as images are uploaded, depositing the embeddings resulting from the service execution in the storage system that will trigger the next step, the Embedding Matrix Generator (EMG).

3.3. Embedding Matrix Generator (EMG)

The Drift Detection Service (DDS) component (as will be explained in Section 3.4) requires a batch of n samples, where $n > 1$, to perform the covariate drift detection process. Therefore, the purpose of the EMG is to generate such input. This batch is built by concatenating the embedding vectors previously generated by the DRS, resulting in a $\hat{x} \in \mathbb{R}^{n \times k}$ matrix. The EMG is, therefore, an intermediate internal service that is executed in response to the events produced by the output of the DRS. It is configured with a specific threshold N, where $N \geq n$, corresponds to the minimum batch size of samples needed to trigger the detection. Consequently, EMG waits until the threshold of N has been reached to build the embedding matrix and provide the DDS with the expected input to perform the final drift detection.

The input bucket of the service contains the embeddings resulting from the DRS component. From the moment the embeddings are deposited, the EMG services are executed. When N embeddings are found in the input bucket, a new object containing a vector with N embeddings is created and deposited in the output bucket. Once this new file is created, all files in the input bucket that triggered the EMG are deleted to ensure the proper functioning of future system executions.

3.4. Drift Detection Service (DDS)

The last pipeline step is the Drift Detection Service (DDS), the final service that checks if covariate drift occurs. This service contains a covariate drift detector fitted with samples from a reference distribution used to train the model.

Covariate drift detectors can be categorized into those based on distance and those using statistical tests. The former calculates the distance between a reference dataset and a data sample to determine if covariate drift is taking place based on a p-value [48], typically established using permutation tests [49]. Common methods in this category include Maximum Mean Discrepancy (MMD) [50], Earth Mover's Distance (EMD) [51], Bhattacharyya Distance [52], Hellinger Distance [53], Population Stability Index (PSI) [54], Jensen-Shannon Divergence [55], Kullback-Leibler Divergence [56], and Histogram Intersection [57]. On the other hand, statistical hypothesis tests [58] directly compute the p-value, including Kolmogorov–Smirnov test [59], Anderson-Darling test [60], Cramér-von Mises test [61], Mann-Whitney U test [62], Welch's t-test [63], and χ^2 test [64].

Regardless of the detector used, the service receives a batch of samples from the EMG and then compares them against the reference samples using the significance level, denoted by α, required for the statistical hypothesis test. In addition, it outputs a boolean value indicating whether drift is occurring, along with the corresponding p-value.

Once the vector with the N embeddings is in the input bucket, the service is invoked, and the result of the drift detection is stored in two output buckets: one within the cluster itself and another where the inference was triggered. This ensures that the user has all the final results of the process in the same cluster, as shown in Figure 2.

4. Evaluation and Results

To evaluate the proposed approach, we established a use-case illustrated in Figure 2 that simulates covariate drift in an

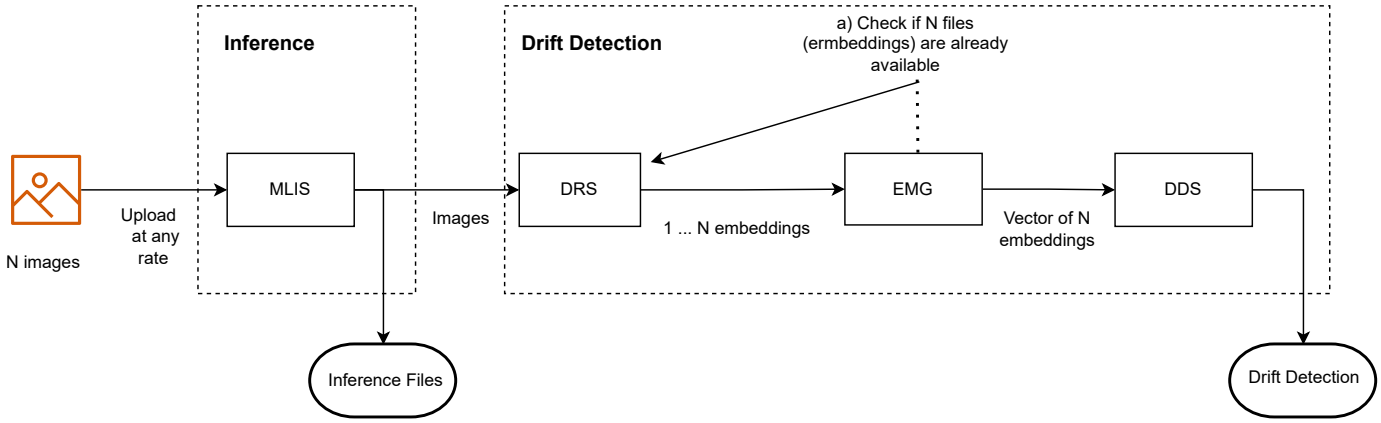


Figure 1: Composition of different serverless functions to provide an ML inference and drift detection service.

edge ML scenario for a classification dataset. On one hand, images can be uploaded to a MinIO bucket to obtain predictions coming from multiple users/clients. These predictions are generated using an edge cluster containing a pre-trained and deployed ML model (MLIS). On the other hand, another cluster is responsible for running the drift detection workflow that reduces the dimensionality (DRS), generates the embedding matrix (EMG), and applies the drift detection method (DDS). This way, we demonstrate the ability of the proposed approach based on the OSCAR framework to span across the edge-to-cloud continuum, including computing resources at the edge of the network (e.g., low-powered devices such as Raspberry Pis) and from Infrastructure as a Service (IaaS) Clouds. Additionally, it involves the combined usage of multiple computer architectures (e.g., arm64 for the edge devices and amd64 for the cloud resources).

4.1. Experimental Setup

This section describes the experimental setup used to evaluate the proposed approach.

4.1.1. Datasets

We use three popular classification datasets for computer vision tasks in the field of machine learning: MNIST [65], Fashion MNIST [66] and CIFAR-10 [67]. Where all of them have 10 different classes or categories.

- **MNIST**: This dataset consists of 28×28 grayscale images (1 channel), with a training set of 60,000 images and a test set of 10,000 images. We split the original training images into subsets: 36,000 and 6,000 images for training and validation of the ML model, 9,000 and 3,000 images for training and validation of the dimensionality reduction model, and 6,000 images as the reference distribution for fitting the covariate drift detector. The remaining 10,000 testing images are used for inference, either with induced covariate drift or without any transformations.
- **Fashion MNIST**: Similar to the MNIST dataset, Fashion MNIST contains 28×28 grayscale images (1 channel). The splitting process follows the same scheme as described for the MNIST dataset.
- **CIFAR-10**: This dataset originally contains 32×32 color images, but we resized them to 28×28, preserving the color (3 channels). This enables us to use the same ML model architecture described in Sections 4.1.3 and 4.1.4 for the dimensionality reduction model. The splitting scheme follows a similar approach to MNIST and Fashion MNIST, with subsets of 30,000 and 5,000 images for training and validation of the ML model, 7,500 and 2,500 images for training and validation of the dimensionality reduction model, and 5,000 images as the reference distribution for fitting the covariate drift detector. As with MNIST and Fashion MNIST, the remaining 10,000 original testing images are used for inference.

4.1.2. Transformations

To simulate covariate drift, certain transformations are applied to the images in each of the described datasets. Three transformations, available in TorchVision [68], are used:

- **GaussianBlur**: This transformation blurs the image using a randomly generated Gaussian blur kernel. The blurring intensity is controlled by the σ parameter, which indicates the standard deviation of the Gaussian kernel.
- **ElasticTransform**: Utilizing displacement vectors based on random offsets, this transformation applies elastic distortions to the image. The strength of the distortions is controlled by the α parameter, while the smoothness of the distortion field is determined by the σ parameter.
- **ColorJitter**: This transformation randomly adjusts the brightness (b), contrast (c), saturation (s), and hue (h) of the image to introduce color variations.

For the MNIST and Fashion MNIST datasets, only GaussianBlur and ElasticTransform are applied, as they consist of

grayscale images. However, all three transformations are applied to the CIFAR-10 dataset, which contains color images suitable for ColorJitter.

It is essential to emphasize that these transformations are applied solely to simulate covariate drift compared to the original dataset. In real-world scenarios, such transformations are not performed, and the data are processed as is.

Each test set contains 10,000 images, as mentioned in Section 4.1.1. These images are either non-transformed (Reference) or have specific transformations applied. For the GaussianBlur transformation test sets, a fixed kernel size of $(5, 9)$ is used, and σ takes values in the range $\{0.25, 0.5, 1.0, 2.0, 4.0\}$ to vary the degree of blurriness. ElasticTransform uses a fixed σ of 5.0, with α ranging from $\{12.5, 25.0, 50.0, 100.0, 200.0\}$. For ColorJitter, brightness (b), contrast (c), and saturation (s) vary in the range $\{0.1, 0.2, 0.4, 0.8, 1.6\}$, while hue ($h$) does so in $\{0.025, 0.05, 0.1, 0.2, 0.4\}$.

4.1.3. ML Model

The MLIS component deploys the same model architecture for inference across all three datasets: MNIST, Fashion MNIST, and CIFAR-10. The architecture consists of a Convolutional Neural Network (CNN) comprising two convolutional layers followed by a Max Pooling operation after each convolutional layer. The output of the CNN is fed into a fully connected layer that outputs predictions for 10 classes. A more detailed view of the architecture is depicted in Figure A.9, and Table A.4 presents the hyperparameter values.

The model is trained for 50 epochs using the AdamW optimizer [69] with a learning rate of 1×10^{-3}. The learning rate is reduced by a factor of 0.1 every 5 epochs if there is no improvement in the validation loss. We retain the weights of the epoch that achieves the best performance on the validation set.

For a comprehensive understanding of the training and validation stages, Figures B.11a, B.12a, and B.13a display the loss curves for MNIST, Fashion MNIST, and CIFAR-10 datasets, respectively.

4.1.4. Dimensionality Reduction Model

To reduce the dimensionality of the input samples, an autoencoder architecture is employed, as explained in Section 3.2. The autoencoder comprises an encoder and a decoder. The encoder consists of three convolutional layers followed by two fully connected layers, reducing the dimensionality to a latent space or embedding of 2 dimensions. Thus, the original image $x \in \mathbb{R}^{28\times28\times1}$ for MNIST and Fashion MNIST, and $x \in \mathbb{R}^{28\times28\times3}$ for CIFAR-10, is transformed into a vector $\hat{x} \in \mathbb{R}^2$. Conversely, the decoder, composed of two fully connected layers and three transposed convolutional layers, reverses the encoding phase to restore the original dimensionality. Both architecture components comprising the autoencoder are detailed in Figure A.10.

Similar to the training phase of the ML model, as explained in Section 4.1.3, the autoencoder is trained for 50 epochs using the same optimizer, learning rate, and learning rate decay strategy. The final weights are obtained using the epoch with the lowest loss on the validation set.

Figures B.11b, B.12b, and B.13b depict the training and validation stages for MNIST, Fashion MNIST, and CIFAR-10, respectively.

4.1.5. Covariate Drift Detector

DDS employs the Maximum Mean Discrepancy (MMD) method [50], a popular technique for multivariate statistical hypothesis testing in two-sample tests [18], along with a Radial Basis Function Kernel (RBF) [70]. Both the detector and the kernel are utilized through the Frouros drift detection library [9]. This detector, in combination with 100 permutations in a permutation test, serves as the multivariate detector responsible for conducting the necessary operations to determine, through p-values, whether the incoming images of the corresponding dataset are experiencing covariate drift or not. To assess the occurrence of covariate drift, a significance level of α[6] = 0.01 is used for each hypothesis test.

4.1.6. Hardware

Two different OSCAR clusters are employed, one utilizing low-powered devices for the edge, and another running on an on-premises Cloud:

- **Edge cluster**: Comprised of four Raspberry Pi 4 model B units, each equipped with 4 GB of RAM and a SoC Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit processor clocked at 1.5 GHz. The $k3s$ minified distribution of Kubernetes is utilized to operate the OSCAR cluster, with one node designated as the front-end and the remaining Raspberry Pis configured as working nodes.

- **On-premises Cloud cluster**: Deployed on an on-premises IaaS (Infrastructure as a Service) Cloud based on OpenStack. The underlying infrastructure consists of 14 Intel Skylake Gold 6130 processors, each with 14 cores, 5.25 TB of RAM, and 2 x 10GbE ports, and 1 Infiniband port per node. A virtual Kubernetes cluster is configured with an interface and a working node, each equipped with eight vCPUs and 32 GB of RAM. This cluster is dynamically provisioned via the Infrastructure Manager (IM)[7][71], an open-source Infrastructure as Code (IaC) tool capable of seamlessly provisioning complex virtualized infrastructure, such as an OSCAR cluster, on multiple Cloud back-ends. The OSCAR cluster comprises a front node with 2 CPUs and 3.7 GB of memory and a working node with 4 CPUs and 3.7 GB of memory.

These cluster configurations provide an efficient and scalable working environment. The on-premises Cloud cluster offers flexibility, allowing the number of nodes to be dynamically adjusted, either manually by the user, through the IM, or by leveraging the underlying elasticity manager in OSCAR. This

[6]Note: This α denotes the significance level of the hypothesis test and is different from the transformation parameter α used in ElasticTransform.

[7]Infrastructure Manager (IM) - `https://im.egi.eu`

manager automatically adds or removes nodes (i.e., virtual machines) based on workload fluctuations, within a range defined by the user at deployment time.

4.2. Covariate Drift Detection

4.2.1. MNIST

The performance of different transformation values on the MNIST test set is compared against a reference test set without any transformation applied. Table 1 presents the results of detecting covariate drift after applying GaussianBlur and ElasticTransform transformations on all MNIST test set images.

For GaussianBlur transformations, the accuracy starts to decrease with $\sigma = 1.0$, and significantly deteriorates for $\sigma = 4.0$, resulting in an accuracy of 0.7448. Drift is detected for all GaussianBlur tests, except for $\sigma = 0.25$, where the transformation is insufficient to impact the model's performance.

In the case of ElasticTransform, different α values result in varying levels of performance loss. From minimal performance reduction with $\alpha = 12.5$ to significant impact with $\alpha = 200.0$, resulting in an accuracy of 0.3799. Drift is detected for each value, except for $\alpha = 12.5$, which undergoes the slightest transformation.

Examples of GaussianBlur transformations are depicted in Figure 3b, where $\sigma = 2.0$ leads to a decrease in accuracy of more than 4%. Similarly, for ElasticTransform with $\alpha = 100.0$, Figure 3c displays samples with a drop in accuracy of over 15%.

In general, as images undergo more severe transformations (higher values of σ for GaussianBlur and α for ElasticTransform), the model's performance is affected to varying degrees.

4.2.2. Fashion MNIST

Table 2 presents the performance of GaussianBlur and ElasticTransform modifications on the Fashion MNIST test set compared to the reference test set. Similarly to the MNIST dataset, the accuracy of GaussianBlur transformations shows a decreasing trend, losing almost 30% in accuracy for $\sigma = 4.0$, reaching 0.6211. Figure 4b illustrates an example of the transformation for $\sigma = 2.0$, where the accuracy drops to 0.6868. The detector detects covariate drift starting from $\sigma = 1.0$, where the accuracy drops more than 6%. Notably, the detector does not detect covariate drift for $\sigma = 0.25$, indicating no significant performance loss, and for $\sigma = 0.5$, where the loss in accuracy is less than 0.5%.

For ElasticTransform, the first two transformations with $\alpha = 12.5$ and $\alpha = 25.0$ are not detected by the detector. The former experiences a 1% drop in accuracy, while the latter loses more than 3%, though it is close to being detected with a p-value of 0.02. The remaining values of α result in a performance decay, leading to an accuracy of 0.6115 for $\alpha = 100.0$ (Figure 4c). Finally, almost 60% in accuracy is lost when $\alpha = 200.0$.

4.2.3. CIFAR-10

Table 3 presents the results for GaussianBlur, ElasticTransform, and ColorJitter transformations applied to the CIFAR-10 test set, comparing them to the reference test set.

For GaussianBlur, none of its variants are detected as covariate drift, as the distance remains relatively constant, ranging from 20.19×10^{-5} to 22.31×10^{-5}. However, the accuracy decreases progressively from $\sigma = 0.5$ with a 1% loss, to over 30% for $\sigma = 4.0$, as depicted in Figure 5b.

Conversely, all ElasticTransform transformations are detected as covariate drift. Even the mildest transformation with $\alpha = 12.5$ results in over a 3% loss in accuracy. Notably, for $\alpha = 100.0$, the accuracy drops by 27%, and for $\alpha = 200.0$, the model's performance decreases by more than 40%, as illustrated in Figure 5c.

In contrast to the grayscale images of MNIST and Fashion MNIST, CIFAR-10's color images undergo ColorJitter transformations. Except for the least invasive case with $b = 0.1, c = 0.1, s = 0.1, h = 0.025$, which is not detected as covariate drift and involves less than a 0.5% loss in accuracy, all other transformations are detected. They result in accuracy reductions ranging from 1% to 30% for the most severe case with $b = 1.6, c = 1.6, s = 1.6, h = 0.4$. As an example, Figure 5d displays the transformation for $b = 0.4, c = 0.4, s = 0.4, h = 0.1$, which involves an accuracy reduction of almost 5%.

4.3. Service Deployment and Execution Time

The services were deployed on each of the OSCAR clusters using the *oscar-cli* tool[8]. To create a serverless data-driven pipeline, the input buckets of the intermediate services are set as the output buckets of the preceding services, as defined in the Functions Definition Language (FDL) file. An excerpt of the FDL file used to deploy the DRS service in OSCAR is shown in Figure 6. Refer to Section 4.4 for access to all the files for reproducibility.

Once all services are deployed, a Python script is responsible for obtaining images from the dataset and uploading them to the MinIO bucket in the Edge cluster, which serves as the entry point to the inference/drift detection system. The upload frequency can be modified in the script.

To simplify experimentation, the dataset tested in Table 1, initially containing 10,000 images per execution method, was reduced to a single test with 100 images (corresponding to a single MNIST digit). The images were uploaded to the input bucket from a computer in another network at a rate of 15 images per minute, taking 6 minutes and 40 seconds for the upload process. At the end of the upload process, 62 MLIS invocations were in pending status due to the Edge cluster's low computing capabilities with Raspberry Pis.

An analysis of the logs for each invocation revealed the execution times of the MLIS service, as shown in Figure 7. The fairly similar time values across invocations indicate minimal variation in the input image. Figure 8 displays the execution times for the drift detection process (DRS and EMG). The last execution of the EMG service stands out, lasting 2 minutes and 21 seconds, due to its involvement in reading all embeddings, creating an array with all data, and deleting all files from the

[8] `https://github.com/grycap/oscar-cli`

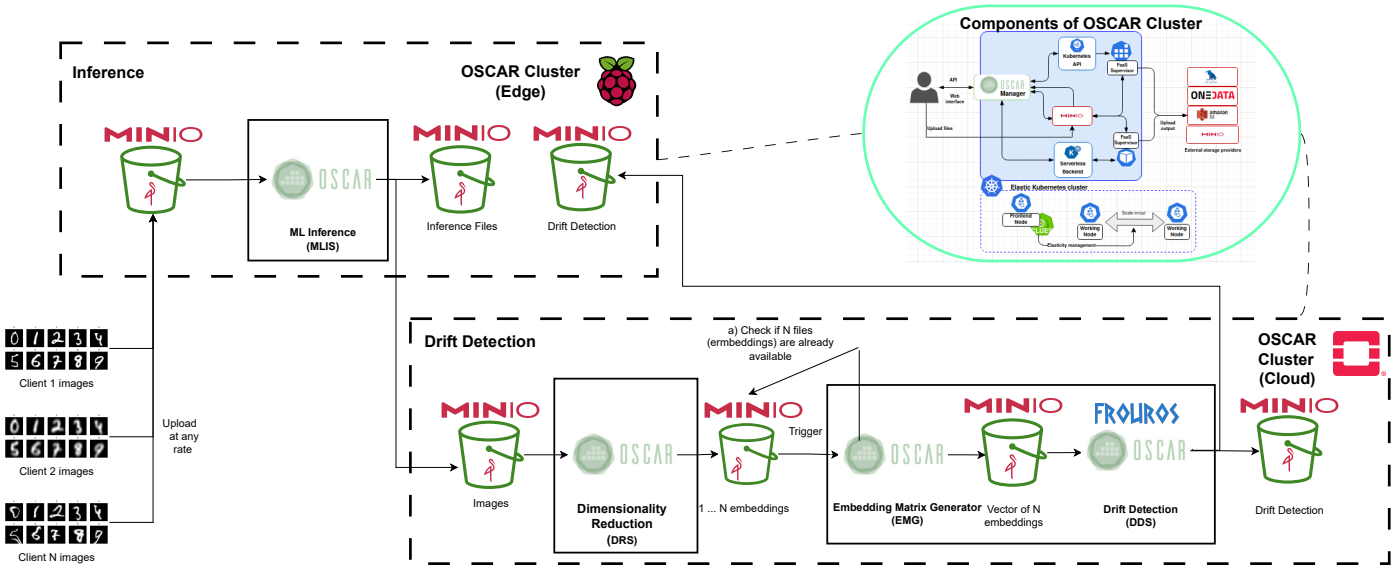


Figure 2: Proposed approach mapped to an edge ML problem. MNIST [65] images are uploaded to a MinIO bucket to perform inference on a Raspberry Pis cluster managed by OSCAR. The dimensionality reduction, embedding concatenation, and covariate drift detection are performed on a different OSCAR cluster.

Method	Parameters	Distance ($\times 10^{-5}$)	p-value	Covariate drift (exist./detect.)	Accuracy
Reference	**-**	**7.35**	**0.16**	**False/False**	**0.9897**
GaussianBlur	$\sigma = 0.25$	7.38	0.16	True/False	0.9897
GaussianBlur	$\sigma = 0.5$	25.88	≤ 0.01	True/True	0.9897
GaussianBlur	$\sigma = 1.0$	200.25	≤ 0.01	True/True	0.9863
GaussianBlur	$\sigma = 2.0$	949.75	≤ 0.01	True/True	0.9467
GaussianBlur	$\sigma = 4.0$	1955.71	≤ 0.01	True/True	0.7448
ElasticTransform	$\alpha = 12.5$	18.93	0.02	True/False	0.9886
ElasticTransform	$\alpha = 25.0$	28.87	≤ 0.01	True/True	0.9847
ElasticTransform	$\alpha = 50.0$	130.34	≤ 0.01	True/True	0.9724
ElasticTransform	$\alpha = 100.0$	740.73	≤ 0.01	True/True	0.8328
ElasticTransform	$\alpha = 200.0$	2622.89	≤ 0.01	True/True	0.3799

Table 1: Results of detecting covariate drift after applying GaussianBlur and ElasticTransform transformations on all MNIST test set images. Appendix Figures C.14 and C.15 show samples for each class after each transformation has been applied respectively.

bucket. The DDS service, invoked only once, requires 13 seconds.

Similar tests were conducted with Fashion MNIST and CIFAR-10 datasets, yielding comparable results. For Fashion MNIST images, the MLIS service averaged 9.57 seconds, and the DRS service averaged 6.15 seconds. For CIFAR-10 images, the MLIS service averaged 9.57 seconds, and the DRS service averaged 6.21 seconds. The EMG service execution did not significantly differ across datasets, as it is independent of image characteristics.

4.4. Reproducibility

To facilitate the replication of the described use cases and reproduce the obtained results, we have made available a GitHub repository[9] containing all the necessary code. This repository includes detailed instructions on setting up the environment and executing the code to replicate the experiments precisely.

[9] `https://github.com/IFCA-Advanced-Computing/serverless-covariate-drift-detection`

5. Conclusions and Future Work

This paper presents a serverless approach for detecting covariate drift in machine learning (ML) models by processing data in batch mode, demonstrating its effectiveness in an edge ML use case for the edge-to-cloud continuum. The approach has shown that it can efficiently detect covariate drift, ensuring robust system performance and reliable predictions through seamless resource provisioning. The successful deployment and operation in an edge ML scenario underscore its potential for broader applicability.

The key contributions of this work include the introduction of a serverless approach to handle batch covariate drift detection, thereby minimizing interference between ML inference and drift detection tasks. This separation ensures that both workflows can be executed efficiently, maintaining the accuracy and reliability of the system.

Future research should extend this approach to handle streaming data. Streaming data presents unique challenges, such as the need for real-time processing and managing data that ar-

Method	Parameters	Distance ($\times 10^{-5}$)	p-value	Covariate drift (exist./detect.)	Accuracy
Reference	**-**	**-4.96**	**0.72**	**False/False**	**0.9104**
GaussianBlur	$\sigma = 0.25$	−4.99	0.72	True/False	0.9104
GaussianBlur	$\sigma = 0.5$	−5.08	0.74	True/False	0.9069
GaussianBlur	$\sigma = 1.0$	13.08	≤ 0.01	True/True	0.8474
GaussianBlur	$\sigma = 2.0$	61.59	≤ 0.01	True/True	0.6868
GaussianBlur	$\sigma = 4.0$	102.31	≤ 0.01	True/True	0.6211
ElasticTransform	$\alpha = 12.5$	−4.41	0.67	True/False	0.9004
ElasticTransform	$\alpha = 25.0$	22.17	0.02	True/False	0.8755
ElasticTransform	$\alpha = 50.0$	188.31	≤ 0.01	True/True	0.8124
ElasticTransform	$\alpha = 100.0$	1141.39	≤ 0.01	True/True	0.6115
ElasticTransform	$\alpha = 200.0$	3730.88	≤ 0.01	True/True	0.3143

Table 2: Results of detecting covariate drift after applying GaussianBlur and ElasticTransform transformations on all Fashion MNIST test set images. Appendix Figures C.16 and C.17 show samples for each class after each transformation has been applied respectively.

Method	Parameters	Distance ($\times 10^{-5}$)	p-value	Covariate drift (exist./detect.)	Accuracy
Reference	**-**	**20.19**	**0.02**	**False/False**	**0.6568**
GaussianBlur	$\sigma = 0.25$	20.19	0.02	True/False	0.6568
GaussianBlur	$\sigma = 0.5$	19.56	0.02	True/False	0.6465
GaussianBlur	$\sigma = 1.0$	18.17	0.04	True/False	0.5635
GaussianBlur	$\sigma = 2.0$	18.36	0.04	True/False	0.4112
GaussianBlur	$\sigma = 4.0$	22.31	0.02	True/False	0.3483
ElasticTransform	$\alpha = 12.5$	110.94	≤ 0.01	True/True	0.6205
ElasticTransform	$\alpha = 25.0$	426.86	≤ 0.01	True/True	0.5811
ElasticTransform	$\alpha = 50.0$	1378.72	≤ 0.01	True/True	0.5101
ElasticTransform	$\alpha = 100.0$	4900.34	≤ 0.01	True/True	0.3851
ElasticTransform	$\alpha = 200.0$	13511.72	≤ 0.01	True/True	0.2415
ColorJitter	$b = 0.1, c = 0.1, s = 0.1, h = 0.025$	5.82	0.24	True/False	0.6547
ColorJitter	$b = 0.2, c = 0.2, s = 0.2, h = 0.05$	38.6	≤ 0.01	True/True	0.6448
ColorJitter	$b = 0.4, c = 0.4, s = 0.4, h = 0.1$	988.57	≤ 0.01	True/True	0.6096
ColorJitter	$b = 0.8, c = 0.8, s = 0.8, h = 0.2$	9990.66	≤ 0.01	True/True	0.4616
ColorJitter	$b = 1.6, c = 1.6, s = 1.6, h = 0.4$	17519.42	≤ 0.01	True/True	0.3508

Table 3: Results of detecting covariate drift after applying GaussianBlur, ElasticTransform, and ColorJitter transformations on all CIFAR-10 test set images. Appendix Figures C.18C.19 and C.20 show samples for each class after each transformation has been applied respectively.

rives continuously and without bounds. Adapting the serverless approach to accommodate these requirements will enable continuous monitoring and covariate drift detection in dynamic, real-time environments, thereby expanding its applicability to a wider range of real-world scenarios.

Another promising direction for future work is to explore the broader applicability of the serverless approach in various aspects of ML model monitoring and management. Beyond covariate drift detection, serverless approach can be advantageous for performance monitoring, model versioning, and anomaly detection. Extending the current approach to encompass these areas will provide a more comprehensive solution for the lifecycle management of ML models.

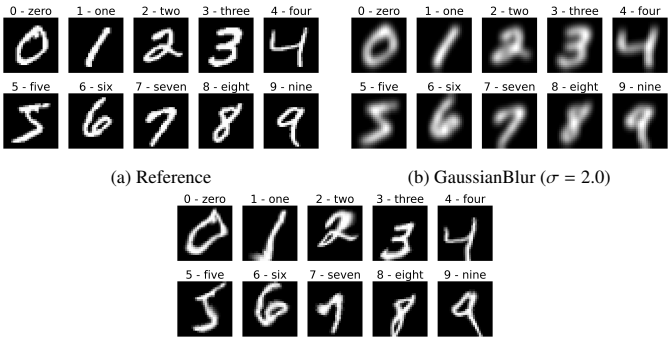
Furthermore, integrating explainability and interpretability features into the serverless approach can significantly enhance model accountability and trustworthiness. By developing techniques to explain the causes of detected drift instances and provide interpretable insights, users can better understand and address the underlying issues. This will not only improve the usability of the system but also facilitate informed decision-making by providing deeper insights into the behavior of ML models.

CRediT Authorship Contribution Statement

JCS: Conceptualization, Software, Validation, Writing - Original Draft. VR: Software, Writing - Original Draft. GM: Conceptualization, Methodology, Resources, Supervision, Funding acquisition, Writing - Original Draft. ALG: Conceptualization, Methodology, Resources, Supervision, Funding acquisition, Writing - Original Draft

Declaration of Competing Interest

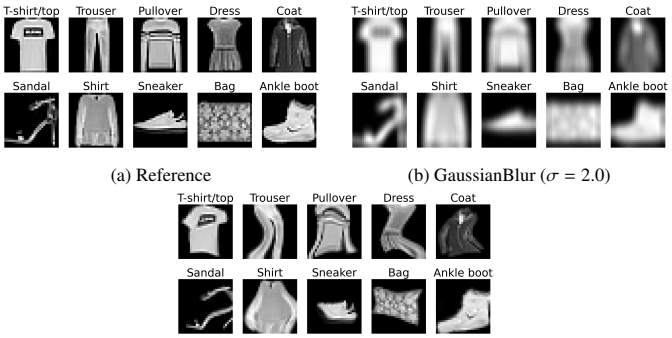
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.



(a) Reference (b) GaussianBlur (σ = 2.0)

(c) ElasticTransform (α = 100.0)

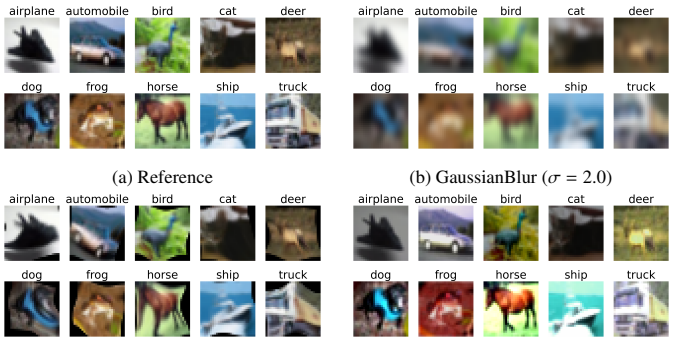
Figure 3: Sample images for each MNIST class. Figure 3a shows a sample without applying any transformation. Figures 3b and 3c apply GaussianBlur and ElasticTransform, respectively, to induce covariate drift.



(a) Reference (b) GaussianBlur (σ = 2.0)

(c) ElasticTransform (α = 100.0)

Figure 4: Sample images for each Fashion MNIST class. Figure 4a shows a sample without applying any transformation. Figures 4b and 4c apply GaussianBlur and ElasticTransform respectively to induce covariate drift.



(a) Reference (b) GaussianBlur (σ = 2.0)

(c) ElasticTransform (α = 100.0) (d) ColorJitter (0.4, 0.4, 0.4, 0.1)

Figure 5: Sample images for each CIFAR-10 class. Figure 5a shows a sample without applying any transformation. Figures 5b, 5c and 5d apply GaussianBlur, ElasticTransform and ColorJitter respectively to induce covariate drift.

```
functions:
  oscar:
    - oscar-cluster1:
        name: dds-service
        memory: 2Gi
        cpu: '2'
        image: ghcr.io/grycap/dds-arm-api
        script: script.sh
        log_level: INFO
        input:
          - storage_provider: minio.default
            path: emg/output
        output:
          - storage_provider: minio.minio-rasp
            path: dds/output
          - storage_provider: minio.default
            path: dds/output
storage_providers:
  minio:
    minio-rasp:
      endpoint:
      region:
      access_key:
      secret_key:
      verify: false
```

Figure 6: Excerpt of the FDL file used to deploy the DRS service in OSCAR.

Data Availability

The data is public and its information is included in the paper text.

Acknowledgements

This work was supported by the project AI4EOSC "Artificial Intelligence for the European Open Science Cloud" that has received funding from the European Union's Horizon Europe Research and Innovation Programme under Grant 101058593. JCS and ALG acknowledge the funding from the Agencia Estatal de Investigación, Unidad de Excelencia María de Maeztu, ref. MDM-2017-0765. GM and VR would like to acknowledge Grant PID2020-113126RB-I00 funded by MICIU/AEI/10.13039/501100011033 and also project PDC2021-120844-I00 funded by MICIU/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR.

References

[1] D. Sun, Standardized serverless ml inference platform on kubernetes, `https://github.com/kserve/kserve`, Online; accessed 13-7-2023 (2019).

[2] T. K. Authors, Machine learning toolkit for kubernetes, `https://github.com/kubeflow/kubeflow`, Online; accessed 13-7-2023 (2018).

[3] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al., Accelerating the machine learning lifecycle with mlflow., IEEE Data Eng. Bull. 41 (4) (2018) 39–45.

[4] C. Yang, S. Sheng, A. Pham, S. Zhao, S. Lee, B. Jiang, F. Dong, X. Guan, F. Ming, BentoML: The framework for building reliable, scalable and cost-efficient AI application.
URL `https://github.com/bentoml/bentoml`

[5] S. Risco, G. Moltó, D. M. Naranjo, I. Blanquer, Serverless workflows for containerised applications in the cloud continuum, Journal of Grid Computing 19 (2021) 30. `doi:10.1007/s10723-021-09570-2`.
URL `https://link.springer.com/10.1007/s10723-021-09570-2`

[6] S. Mäkinen, H. Skogström, E. Laaksonen, T. Mikkonen, Who needs mlops: What data scientists seek to accomplish and how can mlops help?, in: 2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN), IEEE, 2021, pp. 109–112.

[7] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE Transactions on Knowledge and Data Engineering 31 (12) (2018) 2346–2363.

[8] C. a. T. E. C. Directorate-General for Communications Networks, G. ekspertów wysokiego szczebla ds. sztucznej inteligencji, Ethics guidelines for trustworthy AI, Publications Office of the European Union, LU, 2019.

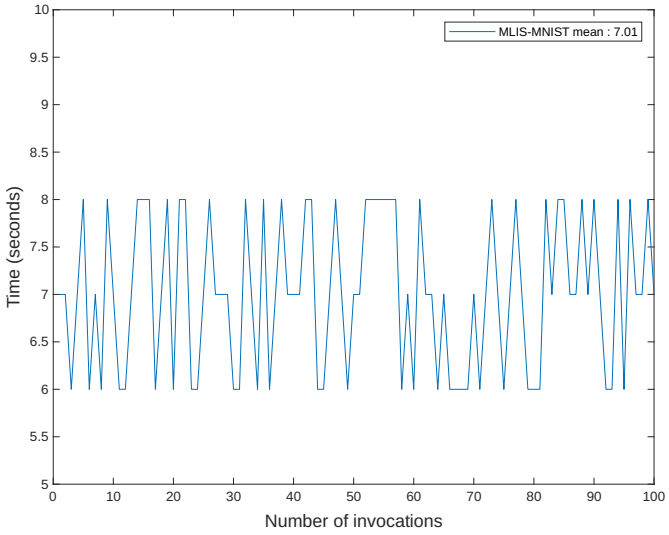


Figure 7: Execution time of the invoked services in the Raspberry Pi cluster.

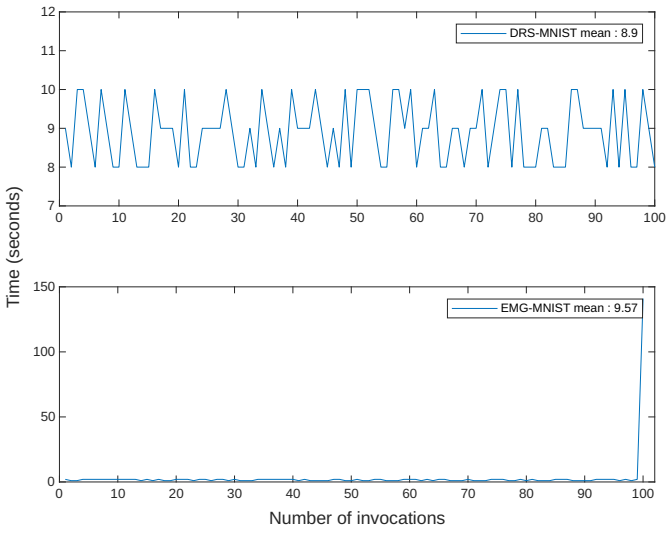


Figure 8: Execution time of the invoked services in the drift detection cluster.

URL https://data.europa.eu/doi/10.2759/346720

[9] J. Céspedes Sisniega, Álvaro López García, Frouros: An open-source python library for drift detection in machine learning systems, SoftwareX 26 (2024) 101733. doi:https://doi.org/10.1016/j.softx.2024.101733.
URL https://www.sciencedirect.com/science/article/pii/S2352711024001043

[10] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, F. Herrera, A unifying view on dataset shift in classification, Pattern recognition 45 (1) (2012) 521–530.

[11] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM computing surveys (CSUR) 46 (4) (2014) 1–37.

[12] H. Shimodaira, Improving predictive inference under covariate shift by weighting the log-likelihood function, Journal of Statistical Planning and Inference 90 (2) (2000) 227–244. doi:https://doi.org/10.1016/S0378-3758(00)00115-4.
URL https://www.sciencedirect.com/science/article/pii/S0378375800001154

[13] A. Tsymbal, The problem of concept drift: definitions and related work, Computer Science Department, Trinity College Dublin 106 (2) (2004) 58.

[14] S. Agrahari, A. K. Singh, Concept drift detection in data stream mining : A literature review, Journal of King Saud University - Computer and Information Sciences 34 (10, Part B) (2022) 9523–9540. doi:https://doi.org/10.1016/j.jksuci.2021.11.006.
URL https://www.sciencedirect.com/science/article/pii/S1319157821003062

[15] M. B. Harries, C. Sammut, K. Horn, Extracting hidden context, Machine learning 32 (2) (1998) 101–126.

[16] R. Klinkenberg, Learning drifting concepts: Example selection vs. example weighting, Intelligent data analysis 8 (3) (2004) 281–300.

[17] H. Raza, G. Prasad, Y. Li, Adaptive learning with covariate shift-detection for non-stationary environments, in: 2014 14th UK Workshop on Computational Intelligence (UKCI), 2014, pp. 1–8. doi:10.1109/UKCI.2014.6930161.

[18] S. Rabanser, S. Günnemann, Z. Lipton, Failing loudly: An empirical study of methods for detecting dataset shift, Advances in Neural Information Processing Systems 32 (2019).

[19] R. S. M. d. Barros, S. G. T. d. C. Santos, An overview and comprehensive comparison of ensembles for concept drift, Information Fusion 52 (2019) 213–244. doi:10.1016/j.inffus.2019.03.006.
URL https://www.sciencedirect.com/science/article/pii/S1566253518308066

[20] J. Zenisek, F. Holzinger, M. Affenzeller, Machine learning based concept drift detection for predictive maintenance, Computers & Industrial Engineering 137 (2019) 106031. doi:https://doi.org/10.1016/j.cie.2019.106031.
URL https://www.sciencedirect.com/science/article/pii/S0360835219304905

[21] C.-C. Lin, D.-J. Deng, C.-H. Kuo, L. Chen, Concept drift detection and adaption in big imbalance industrial iot data using an ensemble learning method of offline classifiers, IEEE Access 7 (2019) 56198–56207. doi:10.1109/ACCESS.2019.2912631.

[22] A. Abbasi, A. R. Javed, C. Chakraborty, J. Nebhen, W. Zehra, Z. Jalil, ElStream: An Ensemble Learning Approach for Concept Drift Detection in Dynamic Social Big Data Stream Learning, IEEE Access 9 (2021) 66408–66419, conference Name: IEEE Access. doi:10.1109/ACCESS.2021.3076264.

[23] K. S. Desale, S. Shinde, N. Magar, S. Kullolli, A. Kurhade, Fake Review Detection with Concept Drift in the Data: A Survey, in: X.-S. Yang, S. Sherratt, N. Dey, A. Joshi (Eds.), Proceedings of Seventh International Congress on Information and Communication Technology, Lecture Notes in Networks and Systems, Springer Nature, Singapore, 2023, pp. 719–726. doi:10.1007/978-981-19-1610-6_63.

[24] A. Van Looveren, J. Klaise, G. Vacanti, O. Cobb, A. Scillitoe, R. Samoilescu, A. Athorne, Alibi detect: Algorithms for outlier, adversarial and drift detection (2019).
URL https://github.com/SeldonIO/alibi-detect

[25] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, T. Seidl, Moa: Massive online analysis, a framework for stream classification and clustering, in: Proceedings of the first workshop on applications of pattern analysis, PMLR, 2010, pp. 44–50.

[26] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, A. Bifet, River: machine learning for streaming data in python, Journal of Machine Learning Research 22 (110) (2021) 1–8.
URL http://jmlr.org/papers/v22/20-1380.html

[27] A. Pérez, G. Moltó, M. Caballer, A. Calatrava, Serverless computing for container-based architectures, Future Generation Computer Systems 83 (2018) 50–59. doi:10.1016/j.future.2018.01.022.
URL https://linkinghub.elsevier.com/retrieve/pii/S0167739X17316485

[28] López García, J. M. De Lucas, M. Antonacci, W. Zu Castell, M. David, M. Hardt, L. Lloret Iglesias, G. Moltó, M. Plociennik, V. Tran, A. S. Alic, M. Caballer, I. C. Plasencia, A. Costantini, S. Dlugolinsky, D. C. Duma, G. Donvito, J. Gomes, I. Heredia Cacha, K. Ito, V. Y. Kozlov, G. Nguyen, P. Orviz Fernández, Z. Šustr, P. Wolniewicz, A cloud-based framework for machine learning workloads and applications, IEEE Access 8 (2020) 18681–18692. doi:10.1109/ACCESS.2020.2964386.

[29] Álvaro López García, Deepaas api: a rest api for machine learning and deep learning models, Journal of Open Source Software 4 (42) (2019) 1517. doi:10.21105/joss.01517.
URL https://doi.org/10.21105/joss.01517

[30] D. M. Naranjo, S. Risco, G. Moltó, I. Blanquer, A serverless gateway for event-driven machine learning inference in multiple clouds, Concurrency and Computation: Practice and Experience n/a (n/a) e6728, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6728. doi:10.1002/cpe.6728.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6728

[31] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, S. Dustdar, Towards a serverless platform for edge {AI}, in: 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), 2019.

[32] V. Muthusamy, A. Slominski, V. Ishakian, Towards enterprise-ready ai deployments minimizing the risk of consuming ai models in business applications, in: 2018 First International Conference on Artificial Intelligence for Industries (AI4I), IEEE, 2018, pp. 108–109.

[33] V. Ishakian, V. Muthusamy, A. Slominski, Serving deep learning models in a serverless platform, in: 2018 IEEE International conference on cloud engineering (IC2E), IEEE, 2018, pp. 257–262.

[34] P. Thinakaran, K. Mahadik, J. Gunasekaran, M. T. Kandemir, C. R. Das, Sandpiper: A cost-efficient adaptive framework for online recommender systems, in: 2022 IEEE International Conference on Big Data (Big Data), IEEE, 2022, pp. 423–430.

[35] H. Zhang, M. Shen, Y. Huang, Y. Wen, Y. Luo, G. Gao, K. Guan, A serverless cloud-fog platform for dnn-based video analytics with incremental learning, arXiv preprint arXiv:2102.03012 (2021).

[36] L. Wang, Y. Zhang, X. Zhu, Concept drift-aware temporal cloud service apis recommendation for building composite cloud systems, Journal of Systems and Software 174 (2021) 110902.

[37] M. Maisenbacher, M. Weidlich, Handling concept drift in predictive process monitoring, in: 2017 IEEE International Conference on Services Computing (SCC), IEEE, 2017, pp. 1–8.

[38] E. Raj, D. Buffoni, M. Westerlund, K. Ahola, Edge mlops: An automation framework for aiot applications, in: 2021 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2021, pp. 191–200.

[39] L. Yang, A. Shami, A lightweight concept drift detection and adaptation framework for iot data streams, IEEE Internet of Things Magazine 4 (2) (2021) 96–101.

[40] A. Mallick, K. Hsieh, B. Arzani, G. Joshi, Matchmaker: Data drift mitigation in machine learning for large-scale systems, in: D. Marculescu, Y. Chi, C. Wu (Eds.), Proceedings of Machine Learning and Systems, Vol. 4, 2022, pp. 77–94.
URL https://proceedings.mlsys.org/paper_files/paper/2022/file/069a002768bcb31509d4901961f23b3c-Paper.pdf

[41] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, O. Tardieu, The serverless trilemma: Function composition for serverless computing, in: Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2017, Association for Computing Machinery, New York, NY, USA, 2017, p. 89–103. doi:10.1145/3133850.3133855.
URL https://doi.org/10.1145/3133850.3133855

[42] M. Verleysen, D. François, The curse of dimensionality in data mining and time series prediction, in: Computational Intelligence and Bioinspired Systems: 8th International Work-Conference on Artificial Neural Networks, IWANN 2005, Vilanova i la Geltrú, Barcelona, Spain, June 8-10, 2005. Proceedings 8, Springer, 2005, pp. 758–770.

[43] L. McInnes, J. Healy, J. Melville, Umap: Uniform manifold approximation and projection for dimension reduction, arXiv preprint arXiv:1802.03426 (2018).

[44] E. Bingham, H. Mannila, Random projection in dimensionality reduction: applications to image and text data, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 2001, pp. 245–250.

[45] K. Pearson, Liii. on lines and planes of closest fit to systems of points in space, The London, Edinburgh, and Dublin philosophical magazine and journal of science 2 (11) (1901) 559–572.

[46] B. Schölkopf, A. Smola, K.-R. Müller, Kernel principal component analysis, in: International conference on artificial neural networks, Springer, 1997, pp. 583–588.

[47] M. A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AIChE journal 37 (2) (1991) 233–243.

[48] R. L. Wasserstein, N. A. Lazar, The asa statement on p-values: context, process, and purpose (2016).

[49] W. J. Welch, Construction of permutation tests, Journal of the American Statistical Association 85 (411) (1990) 693–698.

[50] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, A. Smola, A kernel two-sample test, Journal of Machine Learning Research 13 (25) (2012) 723–773.
URL http://jmlr.org/papers/v13/gretton12a.html

[51] Y. Rubner, C. Tomasi, L. J. Guibas, The earth mover's distance as a metric for image retrieval, International journal of computer vision 40 (2) (2000) 99–121.

[52] A. Bhattacharyya, On a measure of divergence between two multinomial populations, Sankhyā: the indian journal of statistics (1946) 401–406.

[53] E. Hellinger, Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen., Journal für die reine und angewandte Mathematik 1909 (136) (1909) 210–271.

[54] D. Wu, D. L. Olson, Enterprise risk management: coping with model risk in a large bank, Journal of the Operational Research Society 61 (2) (2010) 179–190.

[55] J. Lin, Divergence measures based on the shannon entropy, IEEE Transactions on Information theory 37 (1) (1991) 145–151.

[56] S. Kullback, R. A. Leibler, On information and sufficiency, The annals of mathematical statistics 22 (1) (1951) 79–86.

[57] M. J. Swain, D. H. Ballard, Color indexing, International journal of computer vision 7 (1) (1991) 11–32.

[58] J. Neyman, E. S. Pearson, Ix. on the problem of the most efficient tests of statistical hypotheses, Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character 231 (694-706) (1933) 289–337.

[59] F. J. Massey Jr, The kolmogorov-smirnov test for goodness of fit, Journal of the American statistical Association 46 (253) (1951) 68–78.

[60] F. W. Scholz, M. A. Stephens, K-sample anderson–darling tests, Journal of the American Statistical Association 82 (399) (1987) 918–924.

[61] H. Cramér, On the composition of elementary errors: First paper: Mathematical deductions, Scandinavian Actuarial Journal 1928 (1) (1928) 13–74.

[62] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, The annals of mathematical statistics (1947) 50–60.

[63] B. L. Welch, The generalization of 'student's'problem when several different population varlances are involved, Biometrika 34 (1-2) (1947) 28–35.

[64] K. Pearson, X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 50 (302) (1900) 157–175.

[65] Y. LeCun, The mnist database of handwritten digits, http://yann.lecun.com/exdb/mnist/ (1998).

[66] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).

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

[68] TorchVision maintainers and contributors, TorchVision: PyTorch's Computer Vision library (Nov. 2016).
URL https://github.com/pytorch/vision

[69] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101 (2017).

[70] B. E. Boser, I. M. Guyon, V. N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 144–152.

[71] M. Caballer, G. Moltó, A. Calatrava, I. Blanquer, Infrastructure manager: A tosca-based orchestrator for the computing continuum, Journal of Grid Computing 21 (2023) 51. doi:10.1007/s10723-023-09686-7.
URL https://link.springer.com/10.1007/s10723-023-09686-7

Appendix A. Neural Network Architectures

Appendix A.1. Convolutional Neural Network (Model)

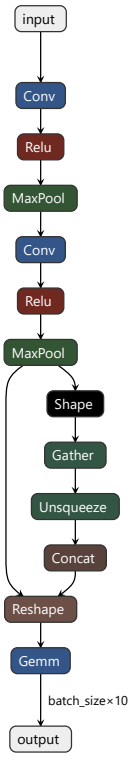


Figure A.9: Convolutional neural network architecture diagram.

Hyperparameter	**Value**
Input Size	[28, 28]
Batch Size	64
Convolutional Layer 1	
In Channels	1 (*grayscale*), 3 (*color*)
Out Channels	16
Kernel Size	5×5
Stride	1
Padding	2
Parameters #	416 (*grayscale*), 1216 (*color*)
Activation	ReLU
Max Pooling 1	
Kernel Size	2×2
Convolutional Layer 2	
In Channels	16
Out Channels	32
Kernel Size	5×5
Stride	1
Padding	2
Parameters #	12832
Activation	ReLU
Max Pooling 2	
Kernel Size	2×2
FC Layer	
Input Size	1568
Output Size	10
Parameters #	15690
Activation	Softmax

Table A.4: Convolutional neural network architecture hyperparameters.

Appendix A.2. Autoencoder (Dimensionality Reduction)

Appendix B. Training and Validation

Appendix B.1. MNIST

Appendix B.2. Fashion MNIST

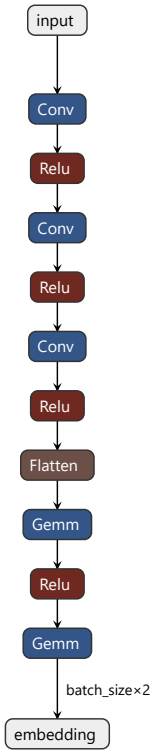
Appendix B.3. CIFAR-10

Appendix C. Dataset Transformations

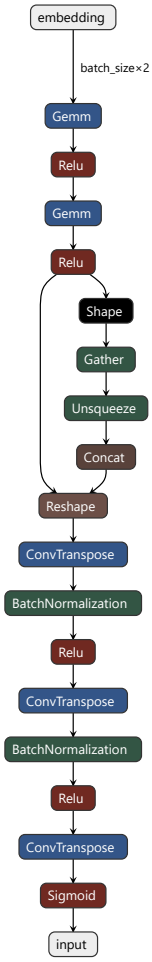
Appendix C.1. MNIST

Appendix C.2. Fashion MNIST

Appendix C.3. CIFAR-10



(a) Encoder



(b) Decoder

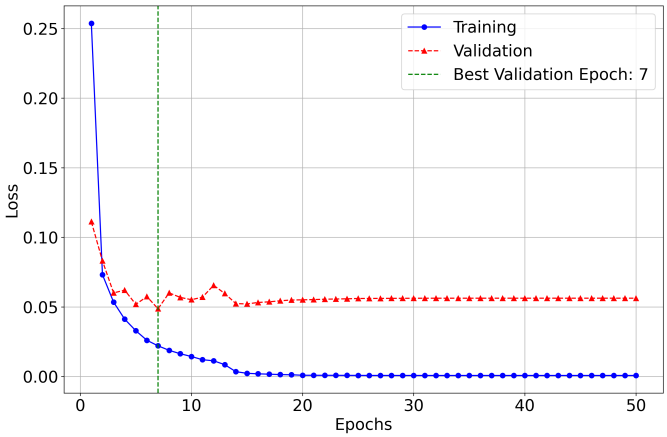
Figure A.10: Autoencoder diagram used for the dimensionality reduction, which consists of an Encoder and a Decoder, depicted in Figures A.10a and A.10b respectively.

Hyperparameter	**Value**
Input Size	[28, 28]
Batch Size	64
Convolutional Layer 1	
In Channels	1 (*grayscale*), 3 (*color*)
Out Channels	8
Kernel Size	3×3
Stride	2
Padding	1
Parameters #	80 (*grayscale*), 220 (*color*)
Activation	ReLU
Convolutional Layer 2	
In Channels	8
Out Channels	16
Kernel Size	3×3
Stride	2
Padding	1
Parameters #	1168
Activation	ReLU
Batch Normalization	
In Channels	16
Parameters #	32
Convolutional Layer 3	
In Channels	16
Out Channels	32
Kernel Size	3×3
Stride	2
Padding	0
Parameters #	4640
Activation	ReLU
FC Layer 1	
Input Size	288
Output Size	128
Parameters #	36992
Activation	ReLU
FC Layer 2	
Input Size	128
Output Size	2
Parameters #	258
Activation	Linear

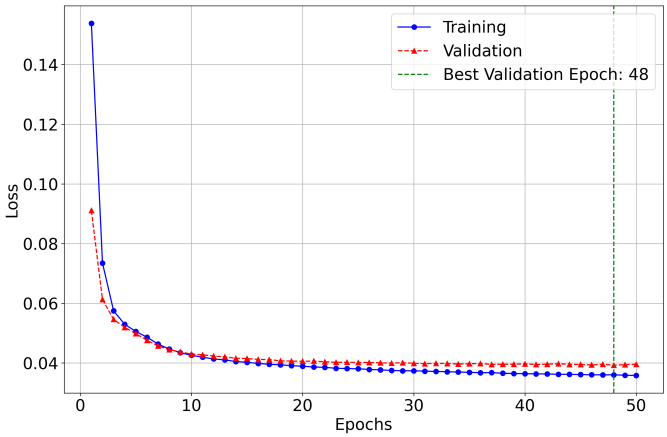
Table A.5: Encoder architecture hyperparameters.

Hyperparameter	Value
Input Size	[2]
Batch Size	64
FC Layer 1	
Input Size	2
Output Size	128
Parameters #	384
Activation	ReLU
FC Layer 2	
Input Size	128
Output Size	288
Parameters #	37152
Activation	ReLU
Convolutional Transpose Layer 1	
In Channels	32
Out Channels	16
Kernel Size	3×3
Stride	2
Padding	0
Output Padding	0
Parameters #	4624
Activation	ReLU
Batch Normalization 1	
In Channels	16
Parameters #	32
Convolutional Transpose Layer 2	
In Channels	16
Out Channels	8
Kernel Size	3×3
Stride	2
Padding	1
Output Padding	1
Parameters #	1160
Activation	ReLU
Batch Normalization 2	
In Channels	8
Parameters #	16
Convolutional Transpose Layer 3	
In Channels	8
Out Channels	1 (*grayscale*), 3 (*color*)
Kernel Size	3×3
Stride	2
Padding	1
Output Padding	1
Parameters #	73 (*grayscale*), 219 (*color*)
Activation	Sigmoid

Table A.6: Decoder architecture hyperparameters.

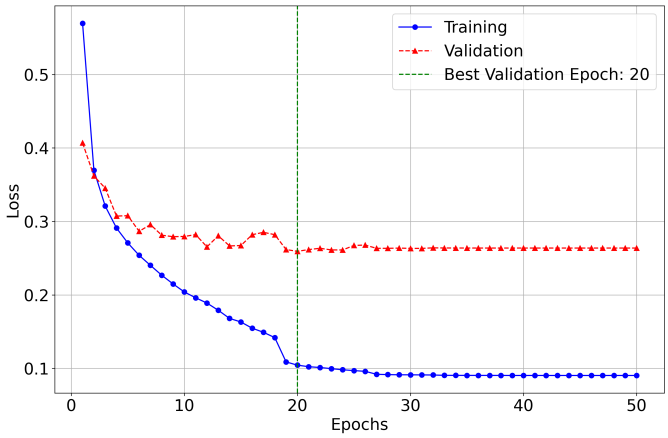


(a) Convolutional neural network (Model)

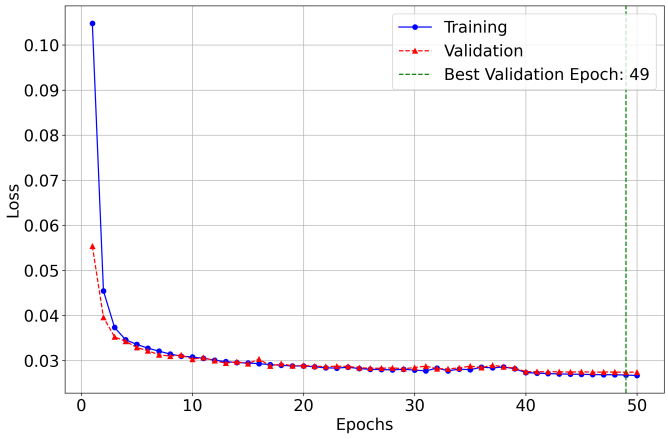


(b) Autoencoder (Dimensionality reduction)

Figure B.11: Training and validation losses for the Convolutional neural network model B.11a and the Autoencoder used for dimensionality reduction B.11b on MNIST.

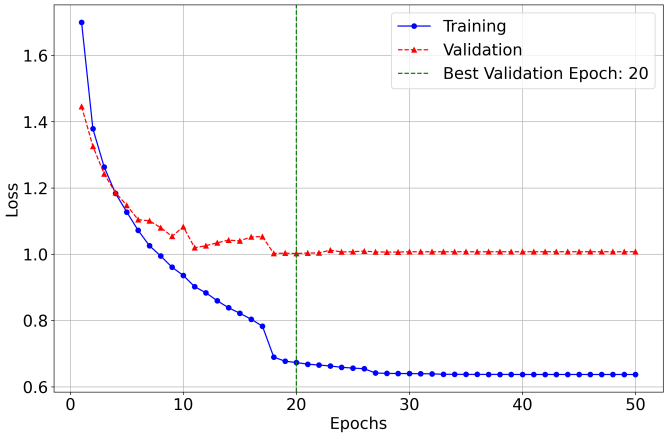


(a) Convolutional neural network (Model)

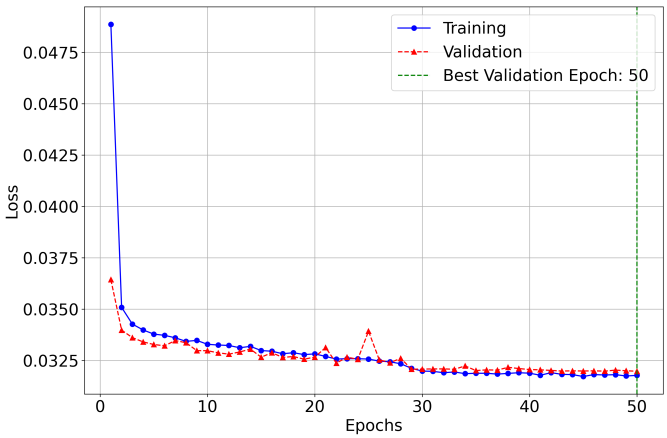


(b) Autoencoder (Dimensionality reduction)

Figure B.12: Training and validation losses for the Convolutional neural network model B.12a and the Autoencoder used for dimensionality reduction B.12b on Fashion MNIST.

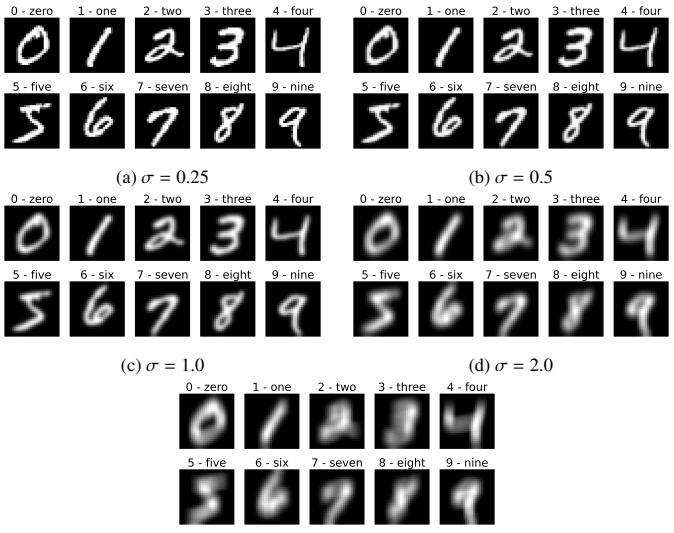


(a) Convolutional neural network (Model)



(b) Autoencoder (Dimensionality reduction)

Figure B.13: Training and validation losses for the Convolutional neural network model B.13a and the Autoencoder used for dimensionality reduction B.13b on CIFAR-10.

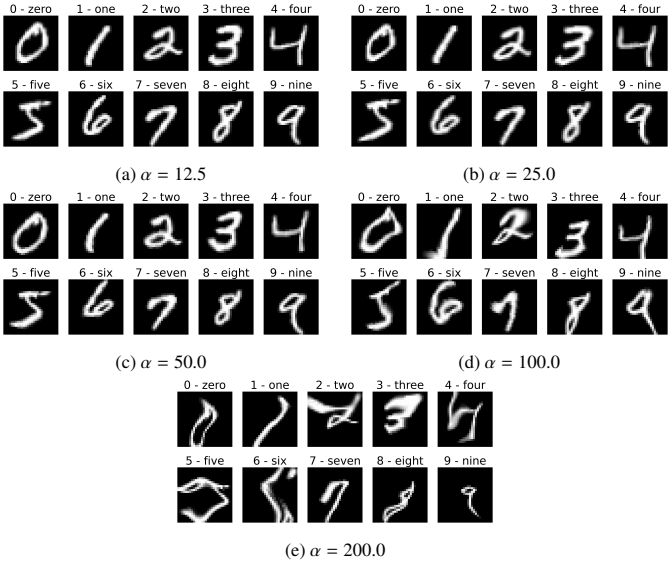


(a) $\sigma = 0.25$ (b) $\sigma = 0.5$

(c) $\sigma = 1.0$ (d) $\sigma = 2.0$

(e) $\sigma = 4.0$

Figure C.14: GaussianBlur transformation applied to a sample of MNIST. Each figure shows a different value of σ.

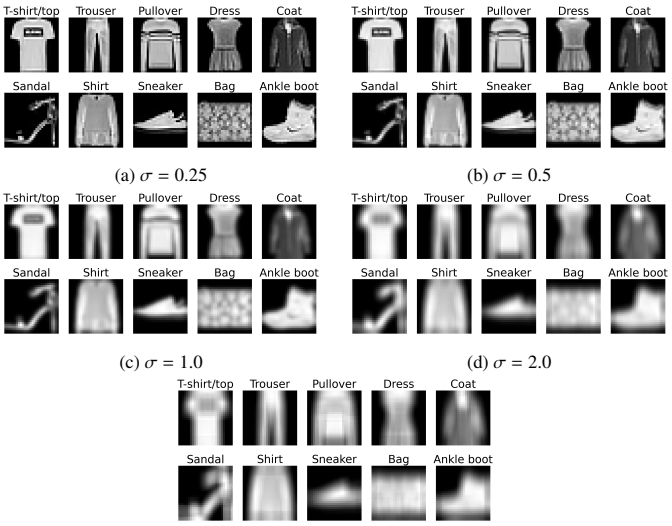


(a) $\alpha = 12.5$ (b) $\alpha = 25.0$

(c) $\alpha = 50.0$ (d) $\alpha = 100.0$

(e) $\alpha = 200.0$

Figure C.15: ElasticTransform transformation applied to a sample of MNIST. Each figure shows a different value of α.

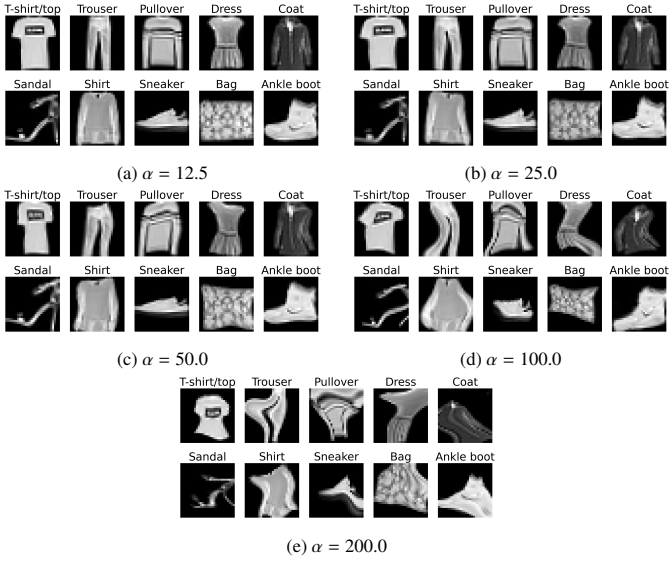


(a) $\sigma = 0.25$ (b) $\sigma = 0.5$

(c) $\sigma = 1.0$ (d) $\sigma = 2.0$

(e) $\sigma = 4.0$

Figure C.16: GaussianBlur transformation applied to a sample of Fashion MNIST. Each figure shows a different value of σ.



(a) $\alpha = 12.5$ (b) $\alpha = 25.0$

(c) $\alpha = 50.0$ (d) $\alpha = 100.0$

(e) $\alpha = 200.0$

Figure C.17: ElasticTransform transformation applied to a sample of Fashion MNIST. Each figure shows a different value of α.

(a) $\sigma = 0.25$ (b) $\sigma = 0.5$

(c) $\sigma = 1.0$ (d) $\sigma = 2.0$

(e) $\sigma = 4.0$

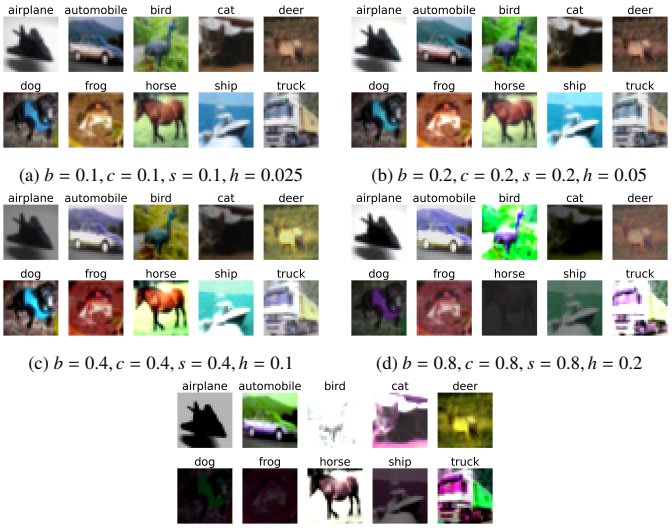
Figure C.18: GaussianBlur transformation applied to a sample of CIFAR-10. Each figure shows a different value of σ.

(a) $\alpha = 12.5$ (b) $\alpha = 25.0$

(c) $\alpha = 50.0$ (d) $\alpha = 100.0$

(e) $\alpha = 200.0$

Figure C.19: ElasticTransform transformation applied to a sample of CIFAR-10. Each figure shows a different value of α.



(a) $b = 0.1, c = 0.1, s = 0.1, h = 0.025$ (b) $b = 0.2, c = 0.2, s = 0.2, h = 0.05$

(c) $b = 0.4, c = 0.4, s = 0.4, h = 0.1$ (d) $b = 0.8, c = 0.8, s = 0.8, h = 0.2$

(e) $b = 1.6, c = 1.6, s = 1.6, h = 0.4$

Figure C.20: ColorJitter transformation applied to a sample of CIFAR-10. Each figure shows a different value of brightness, contrast, saturation, and hue.