Adaptive Neural Networks for Online Domain Incremental Continual Learning

Nuwan Gunasekara[1], Heitor Gomes[1], Albert Bifet[1], and Bernhard Pfahringer[1]

AI Institute, University of Waikato, Hamilton, New Zealand

Abstract. Continual Learning (CL) poses a significant challenge to Neural Network (NN)s, where the data distribution changes from one task to another. In Online Domain Incremental Continual Learning (ODICL), this distribution change happens in the input space without affecting the label distribution. In order to adapt to such changes, the model being trained risks forgetting previously learned knowledge (stability). On the other hand, enforcing that the model preserves past knowledge will cause it to fail to learn new concepts (plasticity). We propose Online Domain Incremental Networks (ODIN), a novel method to alleviate catastrophic forgetting by automatically detecting the end of a task using concept drift detection. As a consequence, ODIN does not require the specification of task ids. ODIN maintains a pool of NNs, each trained on a single task and frozen for further updates. A Task Predictor (TP) is trained to select the most suitable NN from the frozen pool for prediction. We compare ODIN against popular regularization and replay methods. It outperforms regularization methods and achieves comparable predictive performance to replay methods.

Keywords: Online Domain Incremental Continual Learning

1 Introduction

Though modern Neural Network (NN)s have shown great success in image classification and natural language processing, they assume training data to be Independent and Identically Distributed (IID). Due to this assumption, once confronted with a distribution shift in the input data, the model may undergo costly retraining to preserve old knowledge while adjusting to the new distribution. Without retraining, an NN receiving non-IID data forgets its past knowledge when confronted with a distribution shift. This phenomenon is known as "catastrophic forgetting" in the literature [9,16].

Continual Learning (CL) attempts to minimize this catastrophic forgetting in NNs via replay and regularization methods [16]. Though current replay methods outperform regularization methods in terms of performance, they may not be suitable for situations with memory and privacy constraints on the replay buffer[16,2]. Even though offline CL methods have been proposed, current research mainly focuses on online methods to solve catastrophic forgetting in NNs. This allows one to develop continually learning agents which are adaptive but also resilient to catastrophic forgetting.

Online Domain Incremental Continual Learning (ODICL) focuses on online CL models, which learn from one input distribution to another with minimum catastrophic forgetting. Here the class distribution remains the same. There are many practical applications of this scenario in the modern IoT world. For example, one can use an ODICL approach to avoid costly retraining of an X-ray image classification model after a distribution shift in the incoming data due to some hardware changes in the X-ray machine[19]. The same scenario can be valid for many NN models that rely on hardware sensor inputs. Also, on specific ODICL settings, replay approaches may be less preferred due to constraints on having a replay buffer. Mainly these are privacy constraints on the replay buffer[2,16].

Considering the practical importance of non-replay ODICL, this work proposes an ODICL method that alleviates catastrophic forgetting in NNs. It is superior to regularization methods and competitive to replay methods. Here a Convolutional Neural Network (CNN) is trained online. Once confronted with a concept shift, it freezes a copy of the current CNN. Task Predictor (TP) is trained to pick the best CNN from the frozen pool for prediction. This approach is further extended to automatically detect concept shifts in incoming data using a Task Detector (TD) instead of relying on an external task id signal. Experiment results reveal that both the proposed methods, with and without automatic TD, surpass the current popular regularization methods and have competitive performance compared to replay methods.

The main contributions of this paper are the following:

1. Online Domain Incremental Networks (ODIN): we introduce a novel method to alleviate catastrophic forgetting for Online Domain Incremental Continual Learning without using instance replay. Here, a frozen copy of the training CNN is saved in a pool at the end of each task. A Task Predictor is trained to predict the best frozen CNN for evaluation for a given instance. The experiment results reveal that ODIN yields better accuracy than regularization and competitive performance to replay baselines. Furthermore, an in-depth investigation is done to better understand the effectiveness of different TPs on three ODICL datasets.
2. Instead of relying on an external task id signal during training, ODIN uses an automatic Task Detector mechanism to detect tasks in the incoming data. ADaptive sliding WINdow (ADWIN) is used to detect drifts in CNN's loss. An incremental drift in the loss is determined as the end of a task. Furthermore, incremental or decremental drifts in CNN's loss detected by ADWIN allow ODIN to dynamically increase or decrease the learning rate. To the best of our knowledge, this automatic Task Detector with Dynamic Learning-Rate (DL) adjustment for ODICL has not been proposed before.

The rest of the paper is organized as follows. The following section presents the current developments in Online Domain Incremental Continual Learning, including some practical use cases. The next section then presents the proposed ODIN for ODICL. The experiments section explains the experimental setup where the proposed method is compared against popular ODICL methods on

three datasets. It also provides insights into the effectiveness of different Task Predictors. The final section provides conclusions and directions for future research.

2 Related work

The literature has thoroughly documented that an NN receiving non-IID data forgets past knowledge when confronted with a concept shift [9,16]. Continual Learning attempts to continually learn with minimal forgetting of past concepts [9,16]. In ODICL, this learning happens online, and the data stream comprises different concepts (distributions) with the same label distribution [16].

CL algorithms use two popular approaches to avoid catastrophic forgetting in NNs: regularization and replay. Regularization algorithms like Elastic Weight Consolidation (EWC) [9] and Learning without Forgetting (LwF) [13] adjust the weights of the network in such a way that it minimizes the overwriting of the weights for the old concept. EWC uses a quadratic penalty to regularize updating the network parameters related to the past concept. It uses the Fisher Information Matrix's diagonal to approximate the importance of the parameters [9]. EWC has some shortcomings: 1) the Fisher Information Matrix needs to be stored for each task, 2) it requires an extra pass over each task's data at the end of the training [16]. Though different versions of EWC address these concerns [16], [5] seems suitable for online CL by keeping a single Fisher Information Matrix calculated by a moving average. LwF uses knowledge distillation to preserve knowledge from past tasks. Here, the model related to the old task is kept separate, and a separate model is trained on the current task. When the LwF receives data for a new task (X_{new}, Y_{new}), it computes the output ($\hat{Y_{old}}$) from the old model for new data X_{new}. During training, assuming that $\hat{Y_{old}}$ and $\hat{Y_{new}}$ are predicted values for X_{new} from the old model and new model, LwF attempts to minimize the loss: $\alpha L_{KD}(Y_{old}, \hat{Y_{old}}) + L_{CE}(Y_{new}, \hat{Y_{new}}) + R$ [16]. Here L_{KD} is the distillation loss for the old model, and α is the hyper-parameter controlling the strength of the old model against the new one. L_{CE} is the cross-entropy loss for the new task. R is the general regularization term. Due to this strong relation between old and new tasks, it may perform poorly in situations where there is a huge difference between old and new task distributions [16].

Replay methods present a mix of instances from the old and current concepts to the NN based on a given policy while training. This reduces the forgetting as the training instances from the old concepts avoid complete overwriting of past concepts' weights. GDumb [18], Experience Replay (ER) [6], and Maximally Interfered Retrieval (MIR) [1] are some of the most popular CL replay methods. GDumb attempts to maintain a class-balanced memory buffer using instances from the stream. At the end of the task, it trains the model using the buffered instances. ER uses reservoir sampling[20] to sample instances from the stream to fill the buffer. Reservoir sampling ensures that every instance in the stream has the same probability of being selected to fill the buffer. ER uses random sampling to retrieve instances from the memory buffer. Despite its simplicity,

ER has shown competitive performance in ODICL[16]. Five (three buffer and two non-buffer) tricks have been proposed by [4] to improve the accuracy of ER in the Online Class Incremental Continual Learning (OCICL) setting. The buffer tricks are independent buffer augmentation, balanced reservoir sampling, and loss-aware reservoir sampling. The two non-buffer tricks are bias control and exponential learning rate decay. Except for bias control which controls the bias of newly learned classes, these tricks can be used in ODICL to improve the performance of a replay method. MIR uses the same reservoir sampling as ER to fill the memory buffer. However, when retrieving instances from the buffer, it first does a virtual parameter update using the incoming min-batch. Then it selects the top k randomly sampled instances with the most significant loss increases by the virtual parameter update for training. In the online implementation in [16], this virtual update is done on a copy of the NN. Replay Using Memory Indexing (REMIND) [7] takes this approach to another level by storing the internal representations of the instances by the initial frozen part of the network and using a randomly selected set of these internal representations to train the last unfrozen layers of the network. Here, REMIND can store more instances' representations using internal low-dimensional features. In general, these replay approaches are motivated by how the hippocampus in the brain stores and replays high-level representations of the memories to the neocortex to learn from them [7]. The empirical survey by [16] suggests that ER and MIR perform better on ODICL than other online CL methods.

Recent research has focused on using ODICL methods to avoid costly retraining in practical situations where the model is confronted with a concept shift. ODICL has been used in X-ray image classification to avoid costly retraining on distribution shifts due to unforeseen shifts in hardware's physical properties [19]. Also, it has been used to mitigate bias in facial expression and action unit recognition across different demographic groups [8]. Furthermore, ODICL was used to counter retraining on concept shifts for multi-variate sequential data of critical care patient recordings [2]. The authors highlight some replay methods' infeasibility due to strong privacy requirements in clinical settings. This concern is further highlighted in the empirical study in [16].

Most current ODICL methods rely on an explicit end-of-task signal during training. EWC and LwF use this signal to optimize weights, while replay methods can use it to update their replay buffer. However, GDumb, ER, and MIR do not rely on this signal for replay buffer updates. Though [16] defines ODICL as training without the end of the task signal. Implementations such as [8] and [2] use the end of the task signal to employ CL methods such as EWC and LwF. However, on the other hand, the implementation in [19] assumes a gradual distribution shift in the input data distribution where instances from both the new and old tasks can appear in the stream for a certain period.

ODIN comes in two versions. One assumes the presence of an end-of-task signal at training, whereas the other proposes an automated task detection method. When a concept shift is detected, the proposed method freezes a copy of the training NN and adds it into a pool. A predictor is trained to choose the best

network from the frozen pool for a given evaluation instance. As the method avoids a replay buffer, it is a good candidate for settings with higher privacy requirements.

3 Online Domain Incremental Networks

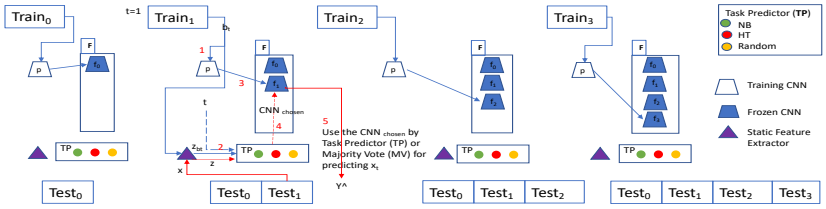


Fig. 1: Proposed ODIN: 1) train network p with incoming mini-batch b_t for t^{th} task, 2) train TP using extracted features and task id, 3) freeze a copy of p at the end of task t 4) at prediction, if enabled, TP predicts CNN_{chosen} via extracted x features 5) predict using CNN_{chosen} or Majority Vote.

In ODICL, the training set is composed of multiple concepts of non-IID data where each concept has a different input distribution with the same label distribution [16]. The goal of the learning algorithm is to minimize catastrophic forgetting of past concepts while performing well on the current concept [16,7]. The initial version of ODIN assumes the availability of the task id at training, which signals the end of a concept to the learning model. However, this information is not available to the model during evaluation. The refined version of ODIN is extended to detect the end of a concept automatically. So, ODIN can be applied to situations where the external task id signal is unavailable at training.

We propose an Online Domain Incremental Networks (ODIN), where CNN p is trained on each concept t with a given Task Predictor (TP). The TPs could be Naive Bayes (NB), or Hoeffding Tree (HT). The TP is trained on mini-batch b_t using extracted features from a feature extractor. Feature extractors extract features from high-dimensional data. Hence, it allows one to use simple learning algorithms on high-dimensional data [10]. Usually, a pre-trained network is used as a feature extractor [10], and its last layer features are used to train the TP. At the end of each task's training, a copy of p is frozen and added to the frozen pool F. Algorithm 1, along with figure 1, further explains this training approach. In ODIN, there are two vote aggregation methods for prediction: Weighted Voting (WV) or votes from the best CNN (CNN_{best}). Weighted Voting uses the TP's probabilities for each frozen CNN as weights. In the CNN_{best} case, it is either selected randomly from the F pool or the one predicted by TP. Algorithm 2 further explains this.

Algorithm 1 ODIN TRAINING ALGORITHM

Input: p: training CNN, F: pool of frozen CNNs, T: task set, X_t: training set for task t, TP: Task Predictor

Initialize pool $F = \{\}$
for all task $t \in T$ **do**
  for all mini-batch b_t in training set X_t for task t **do**
    Train p with the computed the loss L_{b_t} for mini-batch b_t
    if task predictor TP is Naive Bayes or Hoeffding Tree **then**
      $z \leftarrow$ extract features from mini-batch b_t via feature extractor
      TRAIN $TP(z, t)$
    end if
  end for
  Append a copy of p to F
end for

Algorithm 2 ODIN PREDICTION ALGORITHM

Input: x_t: instance of task t, F: pool of frozen CNNs, TP: Task Predictor, useWeightedVoting

$z \leftarrow$ features from instance x_t of task t
if useWeightedVoting **then**
  if TP is Majority Vote **then**
    $votes \leftarrow 1/|F| \sum_{f=1}^{|F|} \text{PREDICT}_f(x_t)$
  else
    $votes \leftarrow 1/|F| \sum_{f=1}^{|F|} \text{PREDICT}_{TP}(z)_f \times \text{PREDICT}_f(x_t)$
  end if
else
  if TP is Random **then**
    Select CNN_{chosen} randomly from pool F
  else
    $\text{CNN}_{chosen} \leftarrow \arg\max_{f \in F} \text{PREDICT}_{TP}(z)$
  end if
  $votes \leftarrow \text{PREDICT}_{\text{CNN}_{chosen}}(x_t)$
end if

Output: $votes$

Algorithm 3 DYNAMIC LEARNING-RATE

Input: lr_0: learning rate at start, d: decay factor($0 < d < 1$), n: instances seen since last drift, $upwardDrift$: whether the estimated loss going up

if $upwardDrift$ **then**
  $lr \leftarrow lr_0 * (1 + d^n)$
else
  $lr \leftarrow lr_0 * (d^n)$
end if

Output: lr

Generally, NN's loss distribution changes when the underlying input distribution changes as the network weights need to be readjusted to match the new distribution. Once a drift in the loss is detected, 1) it would be helpful to learn following the direction of the loss, where the network learns faster if there is an upward drift in the loss, and it learns slower if the drift in the loss is decreasing. Usually, in ODICL, NN's loss gradually decreases for a given task with non-IID training instances. Hence, 2) it would be helpful to reduce the magnitude of the learning for the incoming instances further away from the task's start so that the later instances of the same task do not disturb the learned weights too much. Here we use the drift detector ADWIN [3] to monitor the loss of p. ADWIN uses

Algorithm 4 ODIN TRAINING ALGORITHM WITH DYNAMIC LEARNING-RATE

Input: p: training CNN, F: pool of frozen CNNs, T: task set, X_t: training set for task t, TP: Task Predictor, lr_0: learning rate at start, d: learning rate decay factor

Initialize pool $F = \{\}$
Initialize $n = 0$
Initialize $upwardDrift = true$
for all task $t \in T$ **do**
  for all mini-batch b_t in training set X_t for task t **do**
    Compute the loss L_{b_t} of mini-batch b_t for p
    Update ADWIN$_p$ with L_{b_t}
    if ADWIN$_p$ detects change **then**
      $n \leftarrow 0$
      if change is upward **then**
        $upwardDrift \leftarrow true$
      else
        $upwardDrift \leftarrow false$
      end if
    else
      $n \leftarrow n + 1$
    end if
    if task predictor TP is Naive Bayes or Hoeffding Tree **then**
      $z \leftarrow$ extract features from mini-batch b_t via feature extractor
      TRAIN $TP(z, t)$
    end if
    $lr \leftarrow$ DYNAMIC LEARNING-RATE(lr_0, d, n, $upwardDrift$)
    train p with loss L_{b_t} and lr
  end for
  Append a copy of p to F
end for

exponential histograms for memory efficiency and discards the buffer related to the previous concept once a drift is detected. It also provides an estimation for the mean of the current items in the buffer. Once ADWIN detects a drift in the loss, one can compare the current estimated loss against the previous estimated loss to identify the direction of the loss. This helps determine whether to increase

or decrease the learning rate (point 1). Also, the number of instances seen after the drift can be used to decrease the magnitude of the learning rate (point 2).

The easiest way to manage the learning of a NN is to adjust the learning rate. In order to learn faster for the upward drifts in the loss detected by ADWIN, ODIN increases the learning rate. For the downwards drifts it decreases the learning rate to prevent against large changes of presumably already well-adjusted weights. Also, to decrease the magnitude of the learning in either direction, it uses a decaying factor d^n where d is $0 < d < 1$ and n is the number of instances seen since the last drift. This decaying factor is discussed in [4]. However, they continually decrease the learning rate from the start of learning in their work. Hence it forces NN not to learn too much from instances of later tasks. On the other hand, this Dynamic Learning-Rate (DL) in ODIN allows p to best adjust to the current task. Algorithm 3 and algorithm 4 explain ODIN's Dynamic Learning-Rate adjustment mechanism.

Algorithm 5 ODIN TRAINING ALGORITHM WITH DYNAMIC LEARNING-RATE AND AUTOMATIC TASK DETECTOR

Input: p: training CNN, F: pool of frozen CNNs, T: task set, X_t: training set for task t, TP: Task Predictor, lr_0: learning rate at start, d: learning rate decay factor

Initialize pool $F = \{\}$
Initialize $taskId = 0$
Initialize $n = 0$
Initialize $upwardDrift = true$
for all task $t \in T$ **do**
    for all mini-batch b_t in training set X_t for task t **do**
        Compute the loss L_{b_t} of mini-batch b_t for p
        Update ADWIN_p with L_{b_t}
        if ADWIN_p detects change **then**
            $n \leftarrow 0$
            if change is upward **then**
                $upwardDrift \leftarrow true$
                $taskId \leftarrow taskId + 1$
                Append a copy of p to F
            else
                $upwardDrift \leftarrow false$
            end if
        else
            $n \leftarrow n + 1$
        end if
        if task predictor TP is Naive Bayes or Hoeffding Tree **then**
            $z \leftarrow$ extract features from mini-batch b_t via feature extractor
            TRAIN $TP(z, taskId)$
        end if
        $lr \leftarrow$ DYNAMIC LEARNING-RATE(lr_0, d, n, $upwardDrift$)
        train p with loss L_{b_t} and lr
    end for
end for

Some of the proposed ODICL algorithms rely on an explicit end of the task signal (task id) to identify the start of a new task. The initial ODIN version also relies on explicit task ids to distinguish different tasks for training. This reliance on an explicit task id may preclude one from employing current ODICL algorithms in real-life settings where it may be challenging to identify such a signal explicitly.

One can assume the upward drift in p's loss detected by ADWIN is due to the distribution shift in the underlying input features. Hence, ODIN determines the end of the task when an upward drift is detected. Line 12-15 in algorithm 5 explains this automatic Task Detector (TD) in ODIN. Line 25 of algorithm 5 further integrates DL with this automatic TD. With automatic TD, if the new task is similar to the past task, ADWIN might not detect an upward drift in the loss, as the learning on the new task can improve the prediction of the previous task due to backward knowledge transfer[16]. Hence, detected task ids may not align with actual task ids. Therefore in automatic task detection, the current training network p is included in the F pool only for prediction. In the experiments, the effectiveness of different versions of ODIN were compared against popular regularization and replay baselines.

4 Experiments

The experiments attempt to understand the effectiveness of ODIN against popular online CL baselines. Also, they attempt to identify the effectiveness of Dynamic Learning-Rate adjustments. Furthermore, experiments attempt to identify the effectiveness of the Task Predictor. Lastly, they attempt to identify the effectiveness of ODIN with an automatic Task Detector against online CL baselines that do not use the external end of task signal.

Table 1: Datasets

Dataset	# tasks	instances per train/test task classes	# Channels, H, W
CORe50	11	2000/1000	10	3, 32, 32
RotatedCIFAR10	4	50000/10000	10	3, 32, 32
RotatedMNIST	4	60000/10000	10	1, 28, 28

The experiments were done on three datasets: CORe50 [14], RotatedCIFAR10, and RotatedMNIST. With RotatedCIFAR10 and RotatedMNIST, 90°rotations (0°, 90°, 180°, -90°) of the original images from CIFAR10 [11] and MNIST [12] were considered separate tasks. There were four tasks in each of those two datasets. With CORe50, 11 distinct sessions (8 indoor and 3 outdoor) of the same object were considered as separate tasks: tasks 0-2,4-8 indoor, 3,9, and 10 outdoor. Here the 10 object categories were considered as the class labels. Though it uses the same dataset as in [14] for ODICL, task separation is more natural than the random separation in [14]. Also, our CORe50 version had a

separate evaluation set for each task rather than a mixed evaluation set, as in [14]. This allows one to better understand forgetting in the ODICL setting.

In the Experiments, different versions of ODIN were compared against regularization baselines: LwF, EWC, and replay baselines: ER and MIR. For ER, an extended buffer version was considered in the experiments. Instead of randomly replacing an item from the buffer, we replace an instance from the most represented task's most represented class. It is referenced as ER_{TbCb} in this paper. This ER_{TbCb} is a further extension of [4], where we attempt to balance the buffer with regard to both task and class. This extended ER_{TbCb} was considered so that ODIN without automatic TD can be compared against a good replay method that utilizes the external end of task signal. All the replay methods used a 1k instance buffer. Also, all the methods use a simple CNN (33450 parameters) with four convolution layers. Two types of TPs were used in the experiments for ODIN: NB[1], and HT[1]. Quantized ResNet-18 was used as the feature extractor, and flattened last layer features were used to train the TPs: NB and HT. ResNet was chosen considering [10], where HT was trained on extracted features by the ResNet feature extractor for images. Also, three types of vote aggregation methods were considered in the experiments: Majority Vote (MV), Weighted Voting (WV), and the use of just CNN_{best} just by itself. In the experiments, we also considered a hypothetical scenario of ODIN, where the task id is available at evaluation and is used to determine the correct frozen CNN. This is presented as the "known_{tid}" in the results. It indicates achievable performance if task prediction is perfect. In the results for ODIN, TP_{WV} represents Task Predictor with Weighted Voting, $\text{TP}_{No\text{WV}}$ represents: Task Predictor without Weighted Voting, $\text{TP}^{\text{DL}}_{\text{WV}}$ represents: Task Predictor with Weighted Voting and Dynamic Learning-Rate, $\text{TP}^{\text{DL}}_{No\text{WV}}$ represents: Task Predictor without Weighted Voting and with Dynamic Learning-Rate, $\text{TP}^{\text{TD}}_{\text{WV}}$ represents: Task Predictor with Weighted Voting and automatic Task Detector and, $\text{TP}^{\text{DLTD}}_{\text{WV}}$ represents: Task Predictor with Weighted Voting, Dynamic Learning-Rate and automatic Task Detector[2]. ODIN Dynamic Learning-Rate used the same learning rate decay factor (0.999995) as in [4]'s continuous learning rate decay. All experiments were run using the Avalanche [15] CL platform. The online buffer implementations of ER and MIR were from [16]. Average accuracy and forgetting defined in [16] were used in the evaluation. All experiments were run three times, and relevant averages and standard deviations were considered in the evaluation.

Table 2 contains the average accuracy and forgetting after training on the last task for each method that uses task ids. Considering the average accuracy ranks, ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$ produces the best results. It also has very little forgetting considering average forgetting ranks. Extended ER_{TbCb} has better average accuracy but lags a bit behind ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$ when considering the average accuracy ranks. Except for ODIN random, all ODIN versions achieve better accuracy than the regularization baselines EWC and LwF. Both of the regularization baselines have quite a high forgetting rate. ODIN NB_{WV} yields better results

[1] Source code github.com/nuwangunasekara/ODIN uses online NB and HT[17]

[2] In the legend of the plots, superscripts and subscripts are in lowercase letters

Table 2: Average accuracy and forgetting after training on the last task (use end of the task signal)

dataset	LwF	EWC	ER_{TbCb}	ODIN							
				known^{*}_{tid}	random	MV	HT_{WV}	NB_{WV}	$\text{NB}^{\text{DL}}_{\text{WV}}$	$\text{NB}_{No\text{WV}}$	$\text{NB}^{\text{DL}}_{No\text{WV}}$
Accuracy											
CORe50	0.44 ± 0.02	0.47 ± 0.03	0.65 ± 0.01	0.69 ± 0.01	0.44 ± 0.00	0.54 ± 0.02	0.63 ± 0.05	0.62 ± 0.01	**0.66** ± 0.01	0.62 ± 0.01	0.65 ± 0.01
RotatedCIFAR10	0.44 ± 0.01	0.42 ± 0.01	0.42 ± 0.02	0.49 ± 0.01	0.37 ± 0.02	**0.45** ± 0.01	**0.45** ± 0.01	0.43 ± 0.02	0.44 ± 0.01	0.40 ± 0.00	0.41 ± 0.00
RotatedMNIST	0.66 ± 0.01	0.52 ± 0.01	**0.84** ± 0.01	0.97 ± 0.00	0.48 ± 0.01	0.65 ± 0.02	0.52 ± 0.01	0.79 ± 0.00	0.80 ± 0.00	0.78 ± 0.01	0.78 ± 0.00
Avg	0.51 ± 0.01	0.47 ± 0.02	**0.64** ± 0.01	0.72 ± 0.01	0.43 ± 0.01	0.55 ± 0.02	0.53 ± 0.02	0.62 ± 0.01	0.63 ± 0.01	0.60 ± 0.00	0.62 ± 0.00
Avg Rank	6.00	7.67	3.67		10.00	5.00	4.67	4.33	**2.33**	6.67	4.67
Forgetting											
CORe50	0.19 ± 0.04	0.15 ± 0.05	-0.01 ± 0.02	0.00 ± 0.00	-0.01 ± 0.01	**-0.03** ± 0.01	0.03 ± 0.04	0.01 ± 0.01	0.01 ± 0.00	0.00 ± 0.01	0.01 ± 0.00
RotatedCIFAR10	0.01 ± 0.02	0.07 ± 0.02	0.04 ± 0.00	0.00 ± 0.00	0.02 ± 0.01	**-0.03** ± 0.01	0.04 ± 0.02	-0.02 ± 0.00	-0.02 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
RotatedMNIST	0.24 ± 0.01	0.60 ± 0.02	0.16 ± 0.01	0.00 ± 0.00	0.20 ± 0.02	0.12 ± 0.02	0.60 ± 0.02	**0.12** ± 0.00	0.12 ± 0.01	0.12 ± 0.00	0.13 ± 0.00
Avg	0.14 ± 0.02	0.27 ± 0.03	0.06 ± 0.01	0.00 ± 0.00	0.07 ± 0.01	**0.02** ± 0.02	0.22 ± 0.02	0.04 ± 0.01	0.03 ± 0.00	0.04 ± 0.00	0.04 ± 0.00
Avg Rank	8.00	9.67	5.67		5.33	**1.67**	8.67	4.00	2.67	4.33	5.00

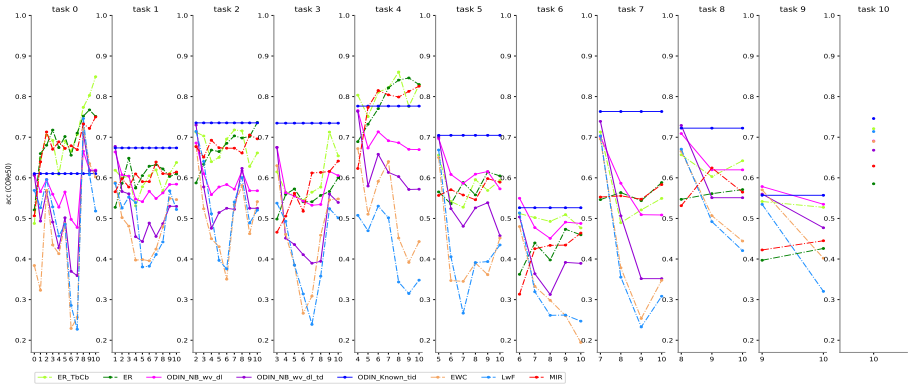
†ER_{TbCb} is ER with an extended task-balanced and class-balanced online buffer.
*known_{tid} is a hypothetical scenario where task id is known at evaluation.

Table 3: Average accuracy and forgetting after training on the last task (do not use end of the task signal)

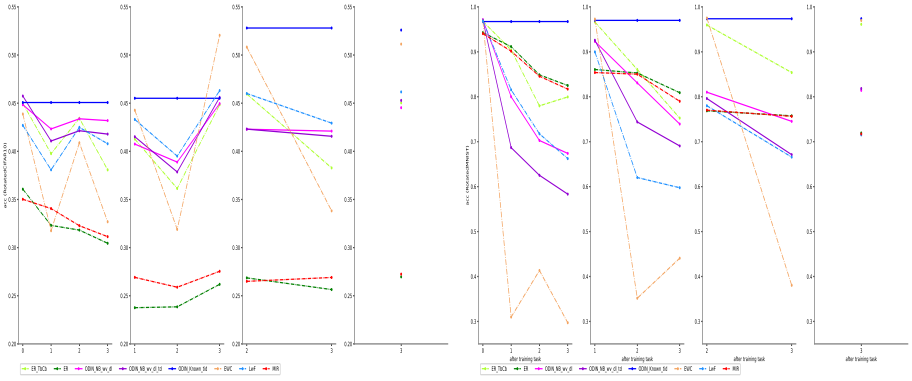
dataset	ODIN		ER	MIR
	$\text{NB}^{\text{TD}}_{\text{WV}}$	$\text{NB}^{\text{DLTD}}_{\text{WV}}$		
Accuracy				
CORe50	0.47 ± 0.04	0.52 ± 0.03	0.61 ± 0.03	**0.62** ± 0.03
RotatedCIFAR10	0.42 ± 0.01	**0.44** ± 0.00	0.27 ± 0.01	0.28 ± 0.01
RotatedMNIST	0.71 ± 0.03	0.69 ± 0.04	**0.78** ± 0.03	0.77 ± 0.01
Avg	0.53 ± 0.02	0.55 ± 0.02	**0.56** ± 0.02	**0.56** ± 0.02
Avg Rank	3.00	2.67	2.33	**2.00**
Forgetting				
CORe50	0.18 ± 0.03	0.17 ± 0.03	-0.09 ± 0.01	**-0.10** ± 0.04
RotatedCIFAR10	0.00 ± 0.01	**0.00** ± 0.00	0.01 ± 0.01	0.01 ± 0.01
RotatedMNIST	0.22 ± 0.04	0.25 ± 0.04	**0.06** ± 0.02	0.07 ± 0.02
Avg	0.14 ± 0.03	0.14 ± 0.03	-0.01 ± 0.02	**-0.01** ± 0.02
Avg Rank	2.67	3.00	2.33	**2.00**

than ODIN $\text{NB}_{No\text{WV}}$ with less average forgetting. This shows that Weighted Voting boosts accuracy. Also, ODIN NB_{WV} yields better accuracy than ODIN HT_{WV} with less forgetting. This shows that NB is a better Task Predictor compared to HT. This is further explored in later experiments. Both $\text{NB}^{\text{DL}}_{\text{WV}}$ and $\text{NB}^{\text{DL}}_{No\text{WV}}$ yields superior accuracy compared to NB_{WV} and $\text{NB}_{No\text{WV}}$. This suggests that Dynamic Learning-Rate improves the overall accuracy. In general, a good Task Predictor, Weighted Voting, and Dynamic Learning-Rate improve ODIN accuracy. Considering the hypothetical ODIN known_{tid} scenario, it is evident that just selecting the correct frozen CNN is sufficient to outperform current baselines by a considerable margin. ODIN known_{tid} also has zero average forgetting across all datasets after training on the last task. This suggests further improvements to the Task Predictors can result in good accuracy gains.

Table 3 only compares ODICL methods that do not use task ids: ODIN $\text{NB}^{\text{TD}}_{\text{WV}}$, ODIN $\text{NB}^{\text{DLTD}}_{\text{WV}}$, ER, and MIR, for a fairer comparison. Here, the ODIN versions use ADWIN as a Task Detector. Also, ER and MIR can be included in the same category as they do not rely on an external task id signal. From the results for this category, it is evident that replay methods slightly outperform ODIN $\text{NB}^{\text{DLTD}}_{\text{WV}}$. Also, replay methods have better forgetting in this category. However, compared to the ODIN methods, they seem to perform quite badly on



(a) CORe50



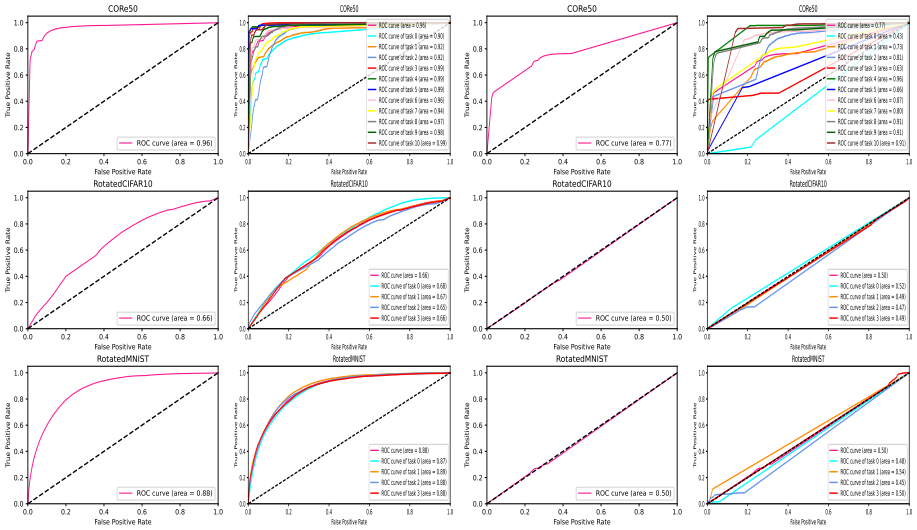
(b) RotatedCIFAR10 (c) RotatedMNIST

Fig. 2: Evaluation accuracy after training on each task

RotatedCIFAR10. Maybe being task aware gives ODIN methods an edge against replay methods on RotatedCIFAR10. Here also, one can see the positive effect of DL on ODIN's accuracy when comparing $\text{NB}_{\text{WV}}^{\text{DLTD}}$ with $\text{NB}_{\text{WV}}^{\text{TD}}$. Considering the results in tables 2 and 3, it is evident that ODIN $\text{NB}_{\text{WV}}^{\text{DL}}$ performs better than ODIN $\text{NB}_{\text{WV}}^{\text{DLTD}}$. This highlights the importance of a good Task Detector. Furthermore, when comparing the two tables, ER_{TbCb} performs better than ER. This highlights the importance of ER to be task aware. In general, when one considers the results of both tables, it is evident that being task aware gives an edge to an ODICL method. Considering the results in both tables, one can conclude that $\text{NB}_{\text{WV}}^{\text{DLTD}}$ performs well in all the datasets.

To get a deeper understanding of each method's predictive performance on old tasks after training on a new task, figure 2a, figure 2b, and figure 2c plot the accuracy of old tasks after training on a new task for selected methods on CORe50, RotatedCIFAR10, and RotatedMNIST datasets. Here top two ODIN methods from each category (with and without the end of the task signal): ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$, ODIN $\text{NB}^{\text{DLTD}}_{\text{WV}}$ were compared against the baselines that use the end of the task signal: EWC, LwF, ER_{TbCb}, and baselines that do not need the end of the task signal: ER and MIR. Hypothetical ODIN known_{tid} is also included in the plots to better understand the upper bound of ODIN's Task Predictor. From figure 2a, it is evident that replay methods perform quite well on past tasks. Especially task-aware ER_{TbCb}. However, their performance has degraded for recent tasks. On the other hand, ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$ has relatively stable performance across all tasks. Hence on average, ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$ performs well on CORe50. This explains its good average accuracy on CORe50 in table 2. Also, ODIN $\text{NB}^{\text{DLTD}}_{\text{WV}}$ has a similar accuracy pattern to ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$. But with less performance. Regularization baselines are quite poor on this dataset. They also have a very high variance. However, as per figure 2b, LwF performs well as ODIN versions on RotatedCIFAR10. Nevertheless, the replay methods ER and MIR perform poorly on that dataset except for ER_{TbCb}. It seems that the learning model needs to be aware of the task identities to perform well on RotatedCIFAR10. As per figure 2c, replay baselines generally perform well on RotatedMNIST. However, except for ER_{TbCb}, the performance gap between ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$ and other replay methods (ER and MIR) seem to narrow for recent tasks. ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$ performed better on the last task than ER and MIR on this dataset. This shows ODIN $\text{NB}^{\text{DL}}_{\text{WV}}$'s ability to perform well on current tasks as well as on past tasks. In all three plots, ODIN with hypothetical TP known_{tid} never forgets after training on a new task. However, it does not improve as well. This explains 0.0 average forgetting for ODIN known_{tid} in table 2.

To further understand the TP's effectiveness, the predicted task id by each TP was compared against the actual task id in non-auto-TD mode against all datasets. This comparison was made for all evaluation instances after training on the last task. Figure 3 shows the ROC curves for the predicted task id and the relevant AUC scores for each TP on each dataset. According to the figure, it is clear that NB is a better Task Predictor for all datasets. This further strengthens the overall strong NB results in table 2. Figure 3b further explains the effectiveness of NB as a TP when predicting each task for a given dataset. From the per-task ROC curves and AUC scores in figures 3b and 3d, it is clear that NB performs similarly on all the tasks for a given dataset. Nevertheless, it does perform slightly better on certain tasks. This is evident in CORe50, with NB performing slightly better for tasks 3,4,5,9, and 10. This generally uniform predictive capability of NB makes it a better Task Predictor than HT.



(a) NB (b) NB per-task (c) HT (d) HT per-task

Fig. 3: Effectiveness of TPs: a & c) micro-average ROC curves for predicted task id and AUC scores, b & d) per-task ROC curves and AUC scores, for TPs NB and HT

5 Conclusion

The proposed ODIN produces competitive results for ODICL in comparison to regularization-based approaches. ODIN with and without automatic Task Detector produces competitive results compared to current popular ODICL baselines without requiring an instance buffer. This makes ODIN a suitable replacement for regularization methods in the ODICL setting. Better Task Predictors, more effective Dynamic Learning-Rate mechanisms, and more responsive Task Detector mechanisms could further improve ODIN's performance in ODICL.

References

1. Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., Tuytelaars, T.: Online continual learning with maximally interfered retrieval. arXiv preprint arXiv:1908.04742 (2019)
2. Armstrong, J., Clifton, D.: Continual learning of longitudinal health records. arXiv preprint arXiv:2112.11944 (2021)
3. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM international conference on data mining. pp. 443–448. SIAM (2007)
4. Buzzega, P., Boschini, M., Porrello, A., Calderara, S.: Rethinking experience replay: a bag of tricks for continual learning. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 2180–2187. IEEE (2021)

5. Chaudhry, A., Dokania, P.K., Ajanthan, T., Torr, P.H.: Riemannian walk for incremental learning: Understanding forgetting and intransigence. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 532–547 (2018)
6. Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P.K., Torr, P.H., Ranzato, M.: On tiny episodic memories in continual learning. arXiv preprint arXiv:1902.10486 (2019)
7. Hayes, T.L., Kafle, K., Shrestha, R., Acharya, M., Kanan, C.: Remind your neural network to prevent catastrophic forgetting. In: European Conference on Computer Vision. pp. 466–483. Springer (2020)
8. Kara, O., Churamani, N., Gunes, H.: Towards fair affective robotics: Continual learning for mitigating bias in facial expression and action unit recognition. arXiv preprint arXiv:2103.09233 (2021)
9. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences **114**(13), 3521–3526 (2017)
10. Korycki, Ł., Krawczyk, B.: Streaming decision trees for lifelong learning. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 502–518. Springer (2021)
11. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
13. Li, Z., Hoiem, D.: Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence **40**(12), 2935–2947 (2017)
14. Lomonaco, V., Maltoni, D.: Core50: a new dataset and benchmark for continuous object recognition. In: Conference on Robot Learning. pp. 17–26. PMLR (2017)
15. Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T.L., De Lange, M., Masana, M., Pomponi, J., Van de Ven, G.M., et al.: Avalanche: an end-to-end library for continual learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3600–3610 (2021)
16. Mai, Z., Li, R., Jeong, J., Quispe, D., Kim, H., Sanner, S.: Online continual learning in image classification: An empirical survey. Neurocomputing **469**, 28–51 (2022). https://doi.org/https://doi.org/10.1016/j.neucom.2021.10.021, https://www.sciencedirect.com/science/article/pii/S0925231221014995
17. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: A multi-output streaming framework. Journal of Machine Learning Research **19**(72), 1–5 (2018), http://jmlr.org/papers/v19/18-251.html
18. Prabhu, A., Torr, P.H., Dokania, P.K.: Gdumb: A simple approach that questions our progress in continual learning. In: European conference on computer vision. pp. 524–540. Springer (2020)
19. Srivastava, S., Yaqub, M., Nandakumar, K., Ge, Z., Mahapatra, D.: Continual domain incremental learning for chest x-ray classification in low-resource clinical settings. In: Albarqouni, S., Cardoso, M.J., Dou, Q., Kamnitsas, K., Khanal, B., Rekik, I., Rieke, N., Sheet, D., Tsaftaris, S., Xu, D., Xu, Z. (eds.) Domain Adaptation and Representation Transfer, and Affordable Healthcare and AI for Resource Diverse Global Health. pp. 226–238. Springer International Publishing, Cham (2021)
20. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) **11**(1), 37–57 (1985)