Project EC4: Trustworthiness by design

Methods and Evaluation tools for Robust AI in Industrial applications

contact@confiance-ai.fr | www.confiance.ai

Document reference: TBA
This report covers the deliverable: TBA

Contributors

	Name	Organization	Role
Co-author	Hatem Hajri	IRT SystemX	co-leading the project and managing the deliverable activities
Co-author	Yassine Tahiri	Airbus	co-leading the project and managing the deliverable activities
Co-author	Pol Labarbarie	IRT SystemX	Chapter B
Co-author	Elies Gherbi	IRT SystemX	Chapter B
Co-author	Mehdi Rezzoug	IRT SystemX	Chapter B
Co-author	Mohamed Ibn Khedher	IRT SystemX	Chapter B
Co-author	Lucas Mattioli	IRT SystemX	Chapter B
Co-author	Martin Gonzalez	IRT SystemX	Chapter C
Co-author	Eliott Py	IRT SystemX	Chapter C
Co-author	Milad Leyli-Abadi	IRT SystemX	Chapter B
Co-author	Luca Mossina	IRT Saint Exupéry	Chapter D
Co-author	Mouhcine Mendil	IRT Saint Exupéry	Chapter D
Co-author	Lucas Hervier	IRT Saint Exupéry	Chapter F
Co-author	Corentin Friedrich	IRT Saint Exupéry	Chapter G
Co-author	Franck Mamalet	IRT Saint Exupéry	Chapter G
Co-author	Eiji Kawasaki	CEA LIST	Chapter E
Co-author	Aurélien Benoit-Lévy	CEA LIST	Chapter E
Co-author	Boussad Addad	THALES	Chapter B
Co-author	Katarzyna Kapusta	THALES	Chapter B
Co-author	Adrien Chan-Hon-Tong	ONERA	Chapter B
Co-author	Stéphane Herbin	ONERA	Chapter B
Co-author	Nelson Fernandez-Pinto	Air Liquide	Chapter C

Document Control

Revision	Date	Commentary	Author
V0.0	2022/08/23	Document initialisation	H. HAJRI (?)
V0.1	2022/09/12	Document setup	M. MENDIL; L. MOSSINA

Contents

Chapter A

Introduction

Addressing the safety and security challenges of complex AI systems is critical to fostering trust in AI. In this context, robustness refers to the ability to withstand or overcome adverse conditions, including digital security risks. Ideally, performance should not deviate significantly. Robustness matters for a number of reasons. First, trust in any tool depends on reliable performance. Trust can degenerate when an ML system performs in an unpredictable way that is difficult to understand. Second, deviation from anticipated performance may indicate important issues that require attention. These issues can include malicious attacks, unmodeled phenomena, undetected biases, or significant changes in data. Ensuring that a model is performing according to its intended purpose under different changes and perturbations is the first priority of the robustness's topic.

In this document, we present the work that has been done during the second year of the Confiance.ai program on the methods and tools related to robustness of artificial intelligence models and their applications to real industrial problems proposed by the program industrial partners. The following document details the methodological progress and numerical results obtained in 6 research and development topics that are strongly interconnected. As an introduction we propose to the reader a short overview of each section.

Environment Alterations of AI models One of the most common approaches to ML watermarking relies on poisoning of the training dataset with specially crafted samples. Such processing embeds multiple legitimate backdoors into the model behaviour that can be later used by the owner to claim the ownership of the watermarked model. First, we implement and test the state-of-the-art techniques. Second, we apply watermarking to the Renault's Welding Inspection use case. We are able to reproduce the results obtained on the public datasets. We analyze and compare three types of watermarking techniques (unrelated, noise, and content). Finally, we test the robustness of the proposed watermarking techniques against a fine-tuning attack.

Robustification methods based on Neural Differential Equations We investigate the problems and challenges of evaluating the robustness of Differential Equation-based (DE) networks against synthetic distribution shifts. We propose a novel and simple accuracy metric which can be used to evaluate intrinsic robustness and to validate dataset corruption simulators. We also

propose methodology recommendations for evaluating different aspects of neural DEs' robustness and comparing them with their discrete counterparts rigorously. We then use this criteria to evaluate an inexpensive data augmentation technique as a reliable way for demonstrating the natural robustness of neural ODEs against simulated image corruptions across multiple datasets. Finally, we provide a solid image purification benchmark with Air Liquide's cylinder counting dataset, a novel corruption-robust system architecture and in depth validation.

Conformal prediction for time series In our chapter we present some methods to apply Conformal Prediction to time series data, on the use case of Demand Forecasting by Air Liquide. *Conformal Prediction* (CP) is an approach to Uncertainty Quantification that does not require special assumptions on the predictive model (neural network, random forests, etc.) nor on the data itself. It can be used in any supervised learning task, such as regression or classification, at the cost of some additional data.

CP yields prediction intervals that are guaranteed to capture the true value of the target at a coverage level (e.g. 90%) specified by the user, according to their operational need. This method can also be applied to "conformalize" en existing model without re-training: the size of the prediction intervals will depend on the uncertainty measured during the calibration process; this can help the user to assess the quality of such predictors on their own data. In our application, the CP procedure is updated online, at each iteration, further improving its empirical performance.

Data Subsampling for Bayesian Neural Networks: A predictive model that is trustworthy and robust is required to be able to provide an uncertainty associated to its prediction. This uncertainty quantification by design is difficult in deep learning as neural networks correspond to strongly non-linear function with a high number of tunable parameters (biases and weights). In this context, Bayesian Neural Networks (BNNs) are able to compute their prediction uncertainty by modeling the distribution over these parameters given the training data set. Despite the fact that BNN cannot be trained by a minimization, we show in this document how to use a noisy gradient descent (Stochastic Gradient Langevin Dynamic) to sample the neural network parameters and thus compute the model prediction uncertainty. In constrast with regular neural networks, we show that naively using sub-sampled mini-batches to train BNN results in poor predictive performances. Going beyond the state of the art, we introduce a corrective term that we call "noise penalty" and show that BNNs recover good predictive performances while trained on mini-batches. In particular, we demonstrate numerical results based on the Air Liquide demand forecast use case. We have developed a software demonstrator that will be delivered and integrated in the confiance.ai environment.

Influence Function for time series

Lipschitz networks for robustness by-design Robustness to L2 perturbations such as adversarial attacks can be addressed by using 1-Lipschitz networks. These networks provide a mean to handle the trade-off between accuracy and robustness with a parametrizable loss function. We propose multiple losses for 1-Lipschitz networks for binary and multiclass classification. We compare the performance of these losses on the Renault Welding use case. In addition to

classification tasks, we also demonstrate how 1-Lipschitz networks can perform very well on semantic segmentation tasks, especially on Valeo Woodscape dataset. Finally, a more theoretical work on orthogonal convolutions is presented. This work was implemented in the deel-lip library in order to provide new capacities to build 1-Lipschitz networks.

Chapter B

Environment Alterations of AI models

Contents

B.1. Introduction

In this study, we investigate the behavior of AI models against environmental alterations. We have studied a set of perturbations that can be applied:

- Evasion attacks consist in perturbing the model inputs by including well calculated noises.
- Patch attacks, a more realistic attack that can printed and positioned on an object or in the environment.

- ML Watermarking consist in perturbing the model behavior by a set of legitimate backdoors in order to enable model identification.
- In addition to these studies, we present DebiAI a web data exploration and visualization tool that allows us to identify dataset biases and errors and compare the model's performance and the results.

B.2. Evaluation against Evasion attacks

The objective of this activity is to evaluate the behavior of neural network models against evasion attacks (adversarial examples). At the beginning of the *confiance.ai* program, a review of the state of the art was launched on adversarial attacks. In this deliverable, we briefly recall the evaluated attacks. As a use case, we have selected the RENAULT use case, which consists on training models for welding quality classification. Regarding the models, we use models from *batch1* that are designed to be robust against environmental perturbation. The rest of this section is organized as follows. First, we recall the principles of some adversarial attacks. Then, we also recall the principle of the robustification methods used in *batch1* to train models. Then, we present the obtained results.

B.2.1 Brief state of the art

In this section, we present a brief state of the art of: 1) evasion attacks and 2) robustification methods. For more information or mathematical details, we invite you to consult the deliverable ISX-EC4-LIV-1498 of *batch1* entitled "Robust & Embeddable Deep Learning by Design".

B.2.1.1 Evasion attacks

Generating an adversarial attack involves adding a small perturbation to the input sample so that the output label is misclassified. Formally, let x be the original input data sample, f be the classifier, and $y = f(x)$ be the label associated with x. A data sample x' is considered an adverse sample of x when x' is close to x under a specific distance metric while $f(x') \neq y$.

Auto-PGD attacks A new technique to identify adversarial attacks was introduced in Croce and Hein [2020] under the name AutoAttack (Auto-PGD). It comprises of a technique to more accurately measure the resilience of the given model. Three issues with traditional PGD attacks that the authors recognized as a driving force behind this method specifically in Croce and Hein [2020]:

- **Fixed step:** Even for difficult problems, solving the maximizing issue with a fixed step size is not ideal problems. Indeed, this value has a big impact on how well the attack works.
- **Budget-agnostic:** Authors demonstrate that there is a loss plateau after a few repetitions. As a result, the number of iterations is not a reliable indicator of the attack's power.

- **Lacking trend awareness:** The method does not take into account how successfully the optimization is evolving and cannot respond.

In APGD (Auto Projected Gradient Descent), the primary idea is to divide the available N_{iter} iterations into two phases: 1) an initial exploration phase in which one searches the feasible set for excellent initial points, and 2) an exploitation phase in which one attempts to maximize the information already collected. The step size is gradually decreased to manage the transition between the two phases. Although the update step in APGD is typical Croce and Hein [2020], this approach differs from traditional PGD in that the step size across iterations is chosen according to the overall budget and the stage of the optimization. Additionally, the maximization resumes from the best point thus far determined once the step size is decreased.

Other attacks *Fast Gradient Sign Method [Goodfellow et al., 2015, FGSM]* creates adversarial examples by adding noise to the original sample along the gradient directions. Two iterative extension of FGSM, namely the *Basic Iterative Method (BIM) by Kurakin et al. [2017]* and the *Projected Gradient Descent (PGD) [Madry et al., 2018a]*, have also been used in the recent literature.

The Jacobian-based Saliency Map Attack (JSMA, Papernot et al. [2016]) generates adversarial examples using forward derivatives (i.e., model Jacobian). JSMA iteratively perturbs features and/or components of the input one at a time, instead of perturbing the whole input to fool the classifier.

Universal Adversarial Perturbations (UAP) Moosavi-Dezfooli et al. [2017] are a special type of untargeted attacks that consist on creating a constant perturbation that successfully misclassifies a specified fraction of the input samples.

Patch attack (PA) Brown et al. [2017b] is a special case of attack that consists on pasting a patch of any shape, generated using a given optimization method (ex: FGSM) on the original input, so as to fool the machine learning model. An adversarial patch attack may be made a universal perturbation too.

DeepFool (DF) Moosavi-Dezfooli et al. [2016] is an untargeted attack based on computing the minimum distance between the original input and the decision boundary.

B.2.1.2 Robustification techniques

Adversarial training Madry et al. [2018b]

It is a methodology for training during conflict. With the use of machine learning to automate critical processes, this approach attempts to provide a comprehensive practical implementation of adversarial training. The objective is to provide several adversarial training techniques, to emphasize the essential techniques and difficulties associated with adversarial training techniques.

The general principle of these methods is to inject noisy data into the Training phase to robustify the model.

Randomized smoothing Lécuyer et al. [2019], Cohen et al. [2019b]

In theses techniques the smoothing is a random function. This approach seeks to create reliable models for regression and classification. It also makes it possible to empirically verify the robustness of the generated model. Randomized smoothing offers verified accuracy for classification tasks up to a given distance from the source of disruptions. It offers an interval for regression tasks where the forecast is assured to be accurate.

B.2.2 Obtained results

The objective of this work is to evaluate the behavior of robust models, trained by robustification techniques, against powerful adversarial attacks such as the auto-pgd attack. The question is, what is the maximum intensity that we can apply on the images. This maximum intensity should be realistic. For this purpose, we have chosen three welding images (the first row of figure B.1) and we have applied three perturbations (the second row of figure B.1) successively of maximum intensity 1%, 5% and 10%. Visually, an intensity of 10% is considered important and we considered not to test disturbances higher than 10%.

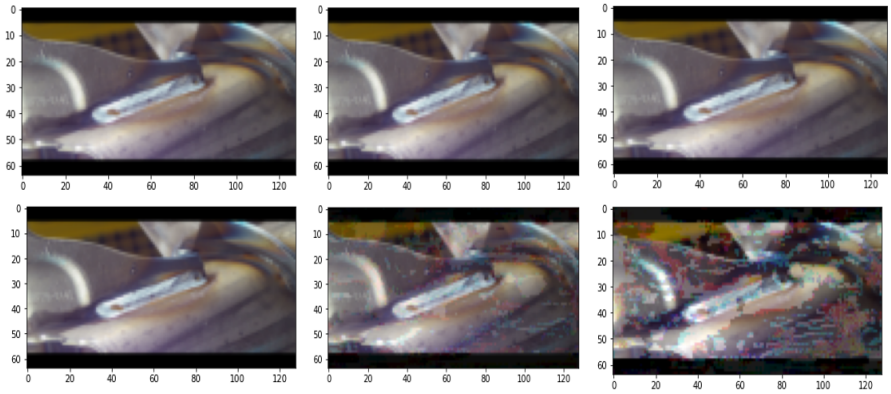


Figure B.1: Image from welding UC under perturbations.

For this purpose, we selected three adversarial attacks (FGSM, PGD, APGD). For each of the attacks, we evaluate maximum intensity values ranging from 1% to 10% with a step of 1%. The tables (B.1, B.2 and B.3) below show the robustness evaluation of models from *batch1* on UC welding, against APGD, PGD and FGSM attacks.
The results show that the best model in terms of robustness against adversarial attacks is the one trained using "Adversarial Training" and "the technique trades". For more investigate the behavior of these two models, we display in the figure B.2 the evolution of the robustness against APGD attack of these two models. Another interesting result is the behavior of the same model against all attacks. In this context, the figures B.4 and B.3 show the behavior of the two selected models against FGSM, PGD and APGD.

Table B.1: Results of A-PGD attacks on robust models

Network	**Metrics**			
	Accuracy (%) before attack	Accuracy (%) attack intensity = 1%	Accuracy (%) attack intensity = 5%	Accuracy (%) attack intensity = 10%
vanilla_Laplace_0.1	98.67	09.93	0.0	0.0
vanilla_Laplace_0.2	99.33	39.73	0.0	0.0
vanilla_Laplace_0.5	98.01	82.11	0.0	0.0
vanilla_Laplace_1.0	80.13	80.13	80.13	80.13
vanilla_Laplace_3.0	80.13	80.13	80.13	80.13
vanilla_Laplace_4.0	80.13	80.13	80.13	80.13
vanilla_Laplace_5.0	80.13	80.13	80.13	80.13
vanilla_Laplace_6.0	80.13	80.13	80.13	80.13
vanilla_Laplace_8.0	80.13	80.13	80.13	80.13
vanilla_Uniform_0.1	80.13	80.13	80.13	80.13
vanilla_Uniform_0.5	95.36	73.50	0.0	0.0
vanilla_Uniform_1.0	80.13	80.13	80.13	80.13
vanilla_Uniform_2.0	80.13	80.13	80.13	80.13
vanilla_Uniform_3.0	80.13	80.13	80.13	80.13
vanilla_Uniform_5.0	80.13	80.13	80.13	80.13
vanilla_Uniform_6.0	80.13	80.13	80.13	80.13
vanilla_Uniform_7.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_0.1	100	29.80	0.0	0.0
vanilla_Gaussian_0.2	99.33	21.85	0.0	0.0
vanilla_Gaussian_0.5	97.35	66.88	0.0	0.0
vanilla_Gaussian_1.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_2.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_3.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_5.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_6.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_7.0	80.13	80.13	80.13	80.13
convnet_trades_epsilon8_beta6	92.71	86.75	18.54	0.66
convnet_adv_training_epsilon8	98.67	96.68	16.55	0.0

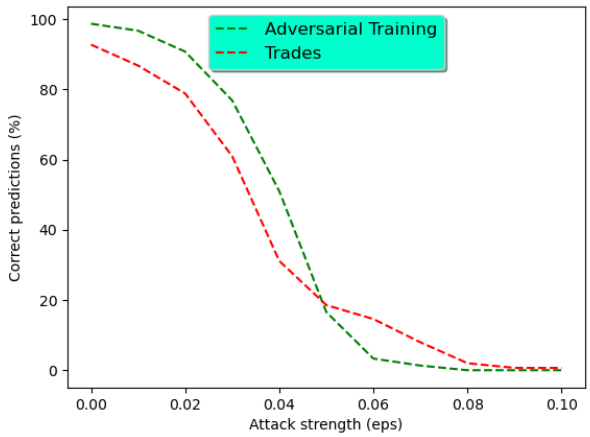


Figure B.2: Behavior of two best models against adversarial attacks (model trained by adversarial training (green curve) and trained by Trades (red curve))

Table B.2: Results of PGD attacks on robust models

Network	**Metrics**			
	Accuracy (%) before attack	Accuracy (%) attack intensity = 1%	Accuracy (%) attack intensity = 5%	Accuracy (%) attack intensity = 10%
vanilla_Laplace_0.1	98.67	09.93	01.32	01.32
vanilla_Laplace_0.2	99.33	39.07	0.66	0.66
vanilla_Laplace_0.5	98.01	84.10	01.98	01.98
vanilla_Laplace_1.0	80.13	80.13	80.13	80.13
vanilla_Laplace_3.0	80.13	80.13	80.13	80.13
vanilla_Laplace_4.0	80.13	80.13	80.13	80.13
vanilla_Laplace_5.0	80.13	80.13	80.13	80.13
vanilla_Laplace_6.0	80.13	80.13	80.13	80.13
vanilla_Laplace_8.0	80.13	80.13	80.13	80.13
vanilla_Uniform_0.1	80.13	80.13	80.13	80.13
vanilla_Uniform_0.5	95.36	78.14	04.63	04.63
vanilla_Uniform_1.0	80.13	80.13	80.13	80.13
vanilla_Uniform_2.0	80.13	80.13	80.13	80.13
vanilla_Uniform_3.0	80.13	80.13	80.13	80.13
vanilla_Uniform_5.0	80.13	80.13	80.13	80.13
vanilla_Uniform_6.0	80.13	80.13	80.13	80.13
vanilla_Uniform_7.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_0.1	100	29.80	0.0	0.0
vanilla_Gaussian_0.2	96.77	11.50	0.0	0.0
vanilla_Gaussian_0.5	97.35	67.54	02.64	02.64
vanilla_Gaussian_1.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_2.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_3.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_5.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_6.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_7.0	80.13	80.13	80.13	80.13
convnet_trades_epsilon8_beta6	92.71	91.39	28.47	07.94
convnet_adv_training_epsilon8	98.67	98.01	24.50	01.32

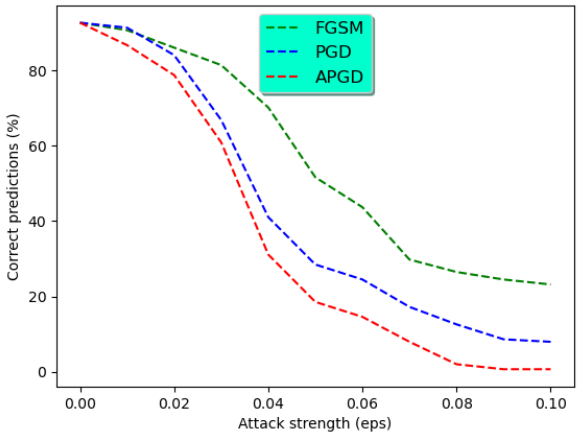


Figure B.3: Behavior of model trained by Trades

Table B.3: Results of FGSM attacks on robust models

Network	**Metrics**			
	Accuracy (%) before attack	Accuracy (%) attack intensity = 1%	Accuracy (%) attack intensity = 5%	Accuracy (%) attack intensity = 10%
vanilla_Laplace_0.1	98.67	02.64	01.32	01.32
vanilla_Laplace_0.2	99.33	58.27	01.32	0.66
vanilla_Laplace_0.5	98.01	90.06	03.31	02.64
vanilla_Laplace_1.0	80.13	80.13	80.13	80.13
vanilla_Laplace_3.0	80.13	80.13	80.13	80.13
vanilla_Laplace_4.0	80.13	80.13	80.13	80.13
vanilla_Laplace_5.0	80.13	80.13	80.13	80.13
vanilla_Laplace_6.0	80.13	80.13	80.13	80.13
vanilla_Laplace_8.0	80.13	80.13	80.13	80.13
vanilla_Uniform_0.1	80.13	80.13	80.13	80.13
vanilla_Uniform_0.5	95.36	83.44	05.29	04.63
vanilla_Uniform_1.0	80.13	80.13	80.13	80.13
vanilla_Uniform_2.0	80.13	80.13	80.13	80.13
vanilla_Uniform_3.0	80.13	80.13	80.13	80.13
vanilla_Uniform_5.0	80.13	80.13	80.13	80.13
vanilla_Uniform_6.0	80.13	80.13	80.13	80.13
vanilla_Uniform_7.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_0.1	100	60.26	0.66	0.0
vanilla_Gaussian_0.2	99.33	37.08	01.32	0.66
vanilla_Gaussian_0.5	97.35	78.80	05.29	03.31
vanilla_Gaussian_1.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_2.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_3.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_5.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_6.0	80.13	80.13	80.13	80.13
vanilla_Gaussian_7.0	80.13	80.13	80.13	80.13
convnet_trades_epsilon8_beta6	92.71	90.72	51.65	23.17
convnet_adv_training_epsilon8	98.67	98.01	72.84	17.21

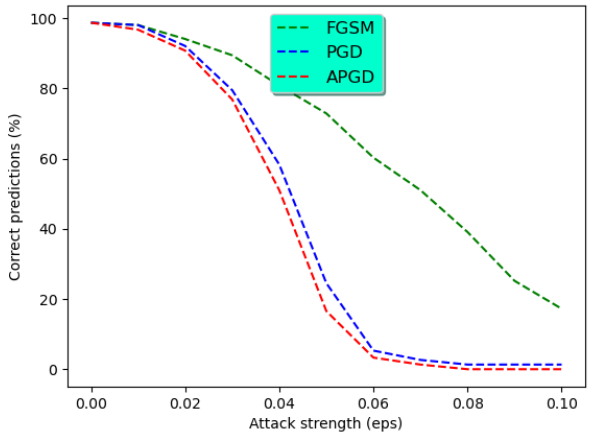


Figure B.4: Behavior of model trained by Adversarial Training

To summarize the results, we can conclude:

- Models trained by adversarial training and trades are more robust than models trained by Randomized smoothing.
- The adversarial attack, APGD, is more powerful than PGD and FGSM.
- On the welding images, applying APGD with 5% of intensity is considered significant as the most robust model loses 74% of its initial performance.

B.3. Patch attacks

B.3.1 Brief state of the art

Although invisible adversarial examples (very slightly modified images) have received strong attention from the computer vision community [Szegedy et al., 2013, Biggio et al., 2013, Kurakin et al., 2017, Madry et al., 2018a, Cohen et al., 2019a], Nguyen et al. [2015] and Brown et al. [2017a] have introduced a new threat which was relatively not considered by the community. This threat, named adversarial patches, are modifications of a small region of an image able to strongly perturb the behavior of deep networks (Fig. B.5).

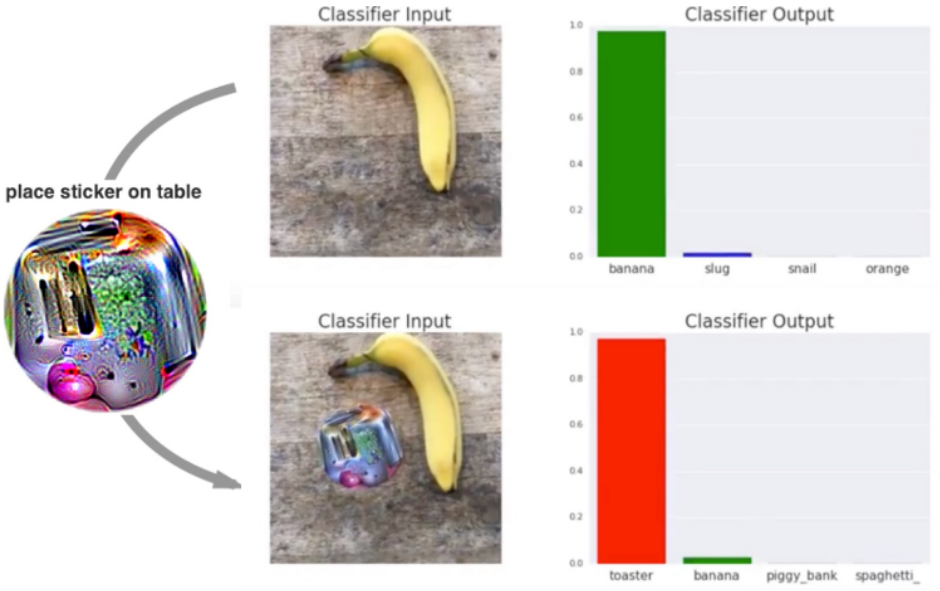


Figure B.5: Figure from Brown et al. [2017a]: a toaster patch is added to a banana image leading the targeted network to predict toaster instead of banana.

Precisely, Nguyen et al. [2015] is not directly related to attacking mechanisms but offers strange images (see figure B.6), which seems to maximize the activation of the network. Adversarial patch attacks tend to have a similar mechanism: the goal is to force a specific dynamic into the network resulting in a very different behavior.

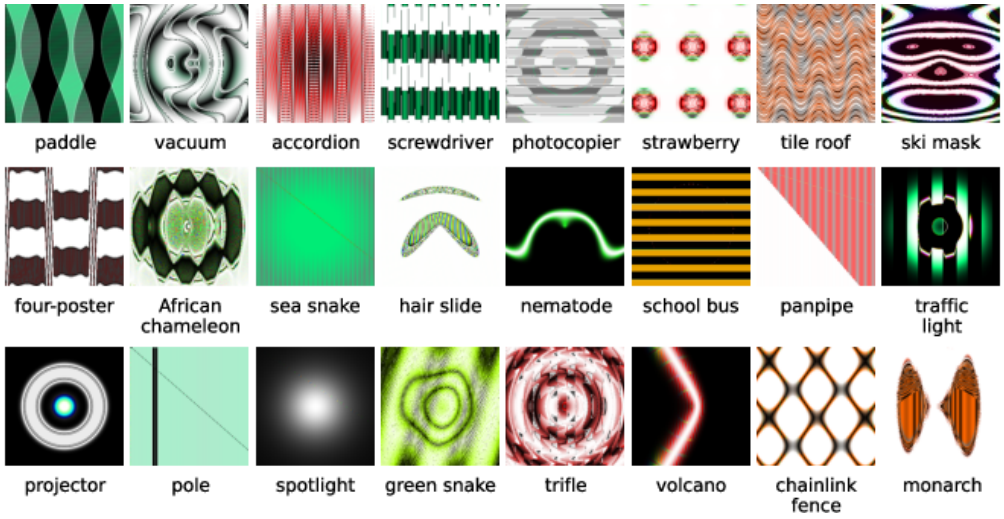


Figure B.6: Strange images from Nguyen et al. [2015] related with strong activation of a targeted deep network.

An important feature from Brown et al. [2017a] is that the patch can be easily created in the physical world. Contrary to invisible adversarial examples, which rely on exact modification at the pixel level, patch attacks rely on a very salient modification which can be more easily constrained to be physically possible. Figure B.6 illustrates the adversarial effect of placing a patch in the scene.

B.3.1.1 From classification to detection

Adversarial patches in the context of classification raise a question: is it an issue that the classifier thinks that there is another object while there is *something*?

This question leads most adversarial patch papers to focus on detection instead of classification: the model is not expected to produce only one label for an image but also a set of labelled bounding boxes. Each box is expected to contain an object with the corresponding label. The mAP (mean Average Precision) usually measures the quality of a detector. The mAP is the mean of the AP (Average Precision) for each class of the dataset. The AP assumes that each produced box has a confidence level. The first step of AP computation is to sort all boxes by decreasing the order of confidence (i.e. most confidence box first). The second step is to loop over all boxes (following the previous order). The current box is compared with the ground truth. If this box overlaps sufficiently[1] with a ground truth box, then it counts as a true positive. Otherwise, it counts as a false alarm. Then, one can remark that each added box can only increase the ratio of ground truth boxes correctly detected (called the recall). But, an added box can either increase or decrease the ratio of good detection over the set of boxes previously considered (called the precision). So, each box corresponds to a new 2D point defined by recall and precision. The

[1]Two boxes are considered as highly overlapping if the area of the intersection over the area of the union (IoU) is below some threshold.

Third and final step to compute the AP is to consider the area under this curve[2]. The AP merges the missing detection and the false alarm to produce a single value summarizing the overall performance.

An essential point of this chapter is that most papers on adversarial patch like Thys et al. [2019] try to decrease the mAP of a detector. Here we are focusing on attacking object detectors, but recently Nesti et al. [2022] introduced a patch for semantic segmentation that produce state-of-the-art attacking results.

B.3.1.2 Main attacks on CNN detectors

The two main papers on attacking CNN detectors are Liu et al. [2018], Lee and Kolter [2019]. Both attacks rely on a kind of unconstrained PGD optimisation [Carlini and Wagner, 2017] on the predefined area of the patch. Training (i.e. optimising the weights) of deep network is done by minimising a loss. Loss of CNN-based detector is usually a mix of:

- an objectness term which forces the model to produce boxes correctly matching the ground truth;
- a cross entropy term for each predicted box which forces the model to label the box according to the ground truth;
- a regression term which improves box localisation.

The only difference between Liu et al. [2018] and Lee and Kolter [2019] is that:

- in Lee and Kolter [2019], the patch is designed by maximising the detection loss with the optimisation variable being the value of the patch. So, the optimisation targets all types of errors (wrong boxes, wrong label or wrong position) to improve the loss.
- In Liu et al. [2018], the adversarial patch is crafted by minimising this detection loss but with a false ground truth (minimising with the correct ground truth will tend to improve performance which is not the hacker goal). This false ground truth discards all true boxes but adds ones directly on the patch. This attack tends to put less emphasis to box localisation and box labels to focus on the production of boxes which is biased toward producing both false positives on the patch and false negatives everywhere else. Note that, initial results from Liu et al. [2018] are plagued by the apparition of *NaN* values into the patch.

B.3.1.3 Attack on transformer

Previous attacks targeted classical CNN. It is now clear that the community is smoothly moving toward transformer-based architectures like ViT [Dosovitskiy et al., 2020] for classification or DETR [Carion et al., 2020] for detection. Using model loss, it was straightforward to try to create patch attacks against these architectures. However, the first attempts of such attacks failed [Shao et al., 2021, Benz et al., 2021, Mahmood et al., 2021]. It has been shown that what looks like

[2]More precisely, it is the area of the closest decreasing function.

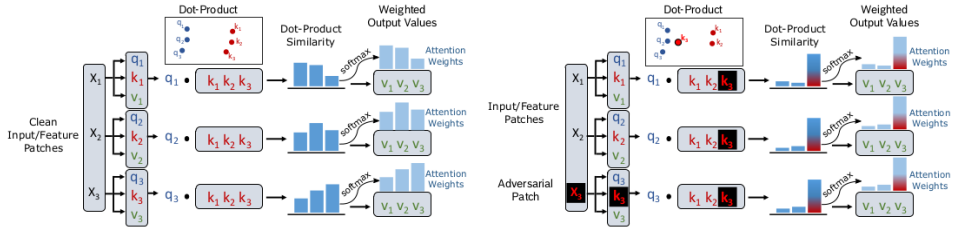


Figure 2. Example of dot-product (self-)attention mechanism for clean (left) and adversarial patch attack (right) settings. Here, q, k, and v stand for projected queries, keys, and value tokens of input features. Left: dot-product attention computes dot-product similarities of a query with all keys, which is later normalized using softmax to obtain per token attention weights. These are multiplied with value tokens to control their contributions in an attention block. Right: *Attention-Fool* losses optimize the adversarial patch in input at X_3 to maximize dot-product similarity of all the queries to the key k_3 (marked in red/black) , which corresponds to moving k_3 closer to the queries cluster. The increase in dot-product similarity of queries with k_3 misdirects the model's attention from image content to adversarial patch.

Figure B.7: Figure from Lovisotto et al. [2022] representing the expected effect of the patch designed to break attention mechanism.

robustness was, in fact, just a kind of gradient masking due to the softmax operation in the inner layer of transformer architecture.

Recently, Lovisotto et al. [2022] showed that transformers are as vulnerable (maybe even more vulnerable) than classical CNN architecture thanks to an attack targeting the attention mechanism of transformers.

For a each head h, an attention head is given by

$$A_h(Q,K,V) = \text{softmax}\left(\frac{W_Q^h Q^h (W_K^h K^h)^T}{\sqrt{d_k}}\right) V^h W_V^h$$

where Q^h, K^h and V^h are the matrices of queries, keys and values respectively, W_Q^h, W_K^h and W_V^h are (learned) projection matrices and $\frac{1}{\sqrt{d_k}}$ a scaled factor. The idea of the attention attack is to optimise the patch such that for each j, $||(W_Q^h Q^h (W_K^h K^h)^T)_{j,i^*}||$ is maximised where i^* correspond to the projected key associated to the patch area. In other words, we want that all queries give exclusive attention to the patch key, as pointed out by figure B.7. To summarise, the goal is to maximise the number of queries that devote their attention to this key. The authors obtained a powerful attack as depicted in figure B.8.

B.3.1.4 Going further in physical implementation: adding the constraint of being stuck on a surface.

The works from previous subsections deal with the passage from a physical patch to a digital patch after sensor acquisition (plus common transformations such as the requirement to be resilient to the point of view change). However, there is a final requirement for deployment: if the patch is a physical object, it should have a physically possible position. The patch can not just float in the air as pointed out by figure B.9. Moreover, the patch should not be too visible otherwise it may be remove by normal users.

In the scope of the *confiance.ai program*, we try to evaluate if the effect of the patch attack was still critical by restraining the patch by this localisation constraint. In particular, we consider

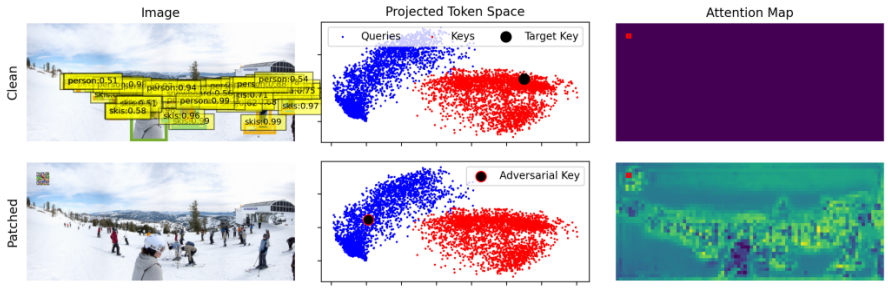


Figure 1. Comparison of clean and adversarially patched input for DETR [8]. The patch shifts a targeted key token towards the cluster of query tokens (middle column). For dot-product attention, this effectively directs the attention of all queries to the malicious token and prevents the model from detecting the remaining objects. The right column compares queries' attention weights to the target key, marked by a red box, between clean and patched inputs and highlights large attention weights drawn by this adversarial key.

Figure B.8: Figure from Lovisotto et al. [2022] representing the critical impact of the offered patch attack.

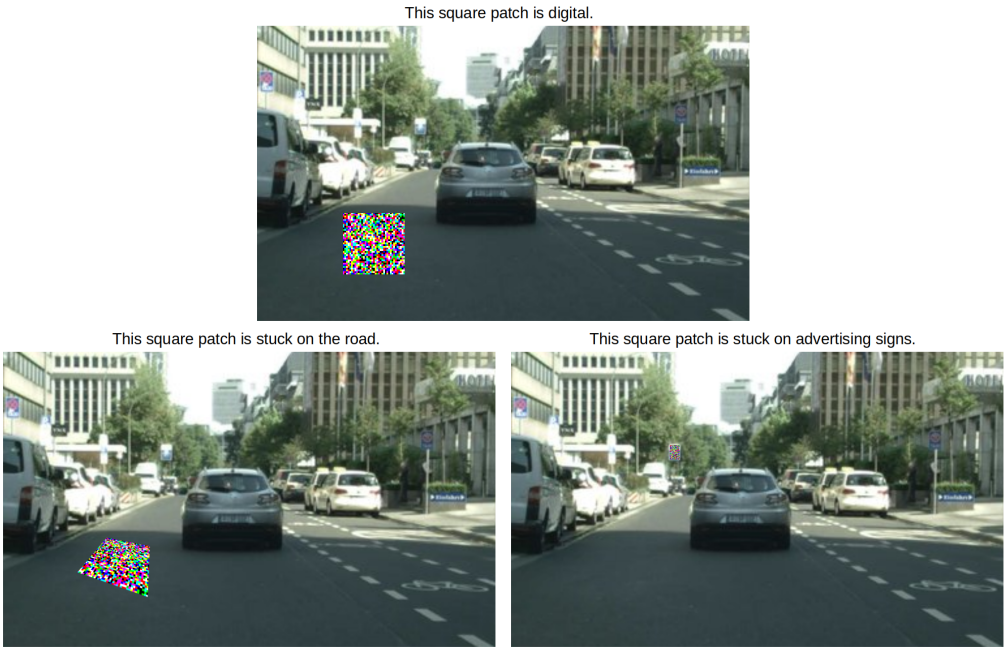


Figure B.9: Taking into account the necessity to embed a physical patch on the scene: the patch can be stuck on the road or on a traffic/ad sign but not just float in the air.

a patch attack in remote sensing data restrained to an individual roof. As remote sensing becomes more and more used by states[3], one can paint on a house roof an adversarial patch. Other users can not remove such patches as the roof may be the property of the hacker (and painting our house roof is usually not prohibited). We take advantage of classical remote sensing datasets like AIRS Chen et al. [2019a] to consider a single roof patch attack. We use the AIRS building footprint ground truth to

- train a semantic segmentation model to predict the building footprint (on one half of the dataset),
- select roof from the other half of the dataset,
- evaluates how a patch attack restricted on this particular roof disturbs the prediction of the network in the neighbourhood.

Figure B.10 illustrates our results.

Preliminary experiments indicated that restricting the attack to a roof (versus using a square with the same area and same centre) has no impact on performance degradation.

However, those experiments should be continued as the current impact is only moderated. We have to determine today if it is due to the nature of the image, the code which computes the patch and/or the fact that segmentation is somehow more robust than detection (which is the main target of patch attack). Nesti et al. [2022] introduced a patch for semantic segmentation that shows encouraging results.

B.3.1.5 Contextual effect

Works from the previous section try to attack different types of architecture or emphasis physical implementation. First works decreasing the mAP of the detector, one can see that their effects are mostly on the patch level. Yet, one may wonder if the effect can be around the patch rather than only on the patch, as pointed out in figure B.11.

Depending on the targeted system, one effect can be more dangerous than another, even at an equivalent mAP level. Suppressing detection only on the patch can be relevant for camouflage purposes and/or to break a surveillance system. Inversely, suppose one considers an emergency steering system in an autonomous car. In that case, it is not in the interest of a potential hacker to stop being detected by the system (because it directly puts the hacker at risk). In this last context, the main threat is if a patch on a wall creates a false negative on the road, i.e., contextual effect.

It is essential to evaluate more precisely than just global mAP decreasing. A pioneering paper on this subject is Saha et al. [2020], which develops attacks and defences for contextual adversarial patches. They introduced the idea of removing false positives on the patch when computing the mAP. They proposed a universal blindness attack targeting one chosen class, an objectness attack, and a targeted attack.

[3] See for example www.lemonde.fr/pixels/article/2022/08/29/experimentee-dans-neuf-departements-la-detection-de-piscines-non-declarees-par-intelligence-artificielle-va-etre-generalisee_6139439_4408996.html

Figure B.10: Illustration of the idea of attacking a physical surface. Here first row represents image, second row represents building footprint, and third row represents patch attack restricted to the central building.

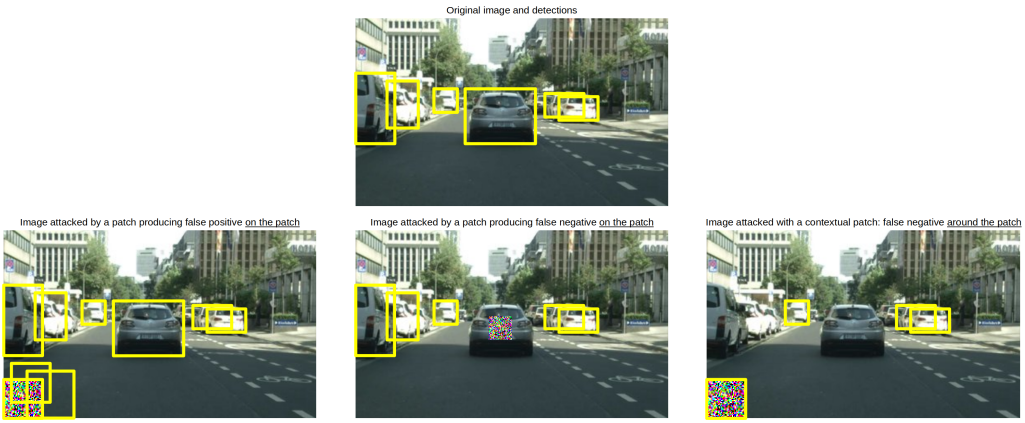


Figure B.11: Above just decreasing the global performance, a patch attack can have qualitatively different kind of effect: mAP can be decreased by adding a lot of false alarms (typically on the patch) or by suppressing detections specifically on the patch (i.e. backdoor and/or immunity idol) or by suppressing detections all around the patch.

B.3.2 Obtained results on welding use case

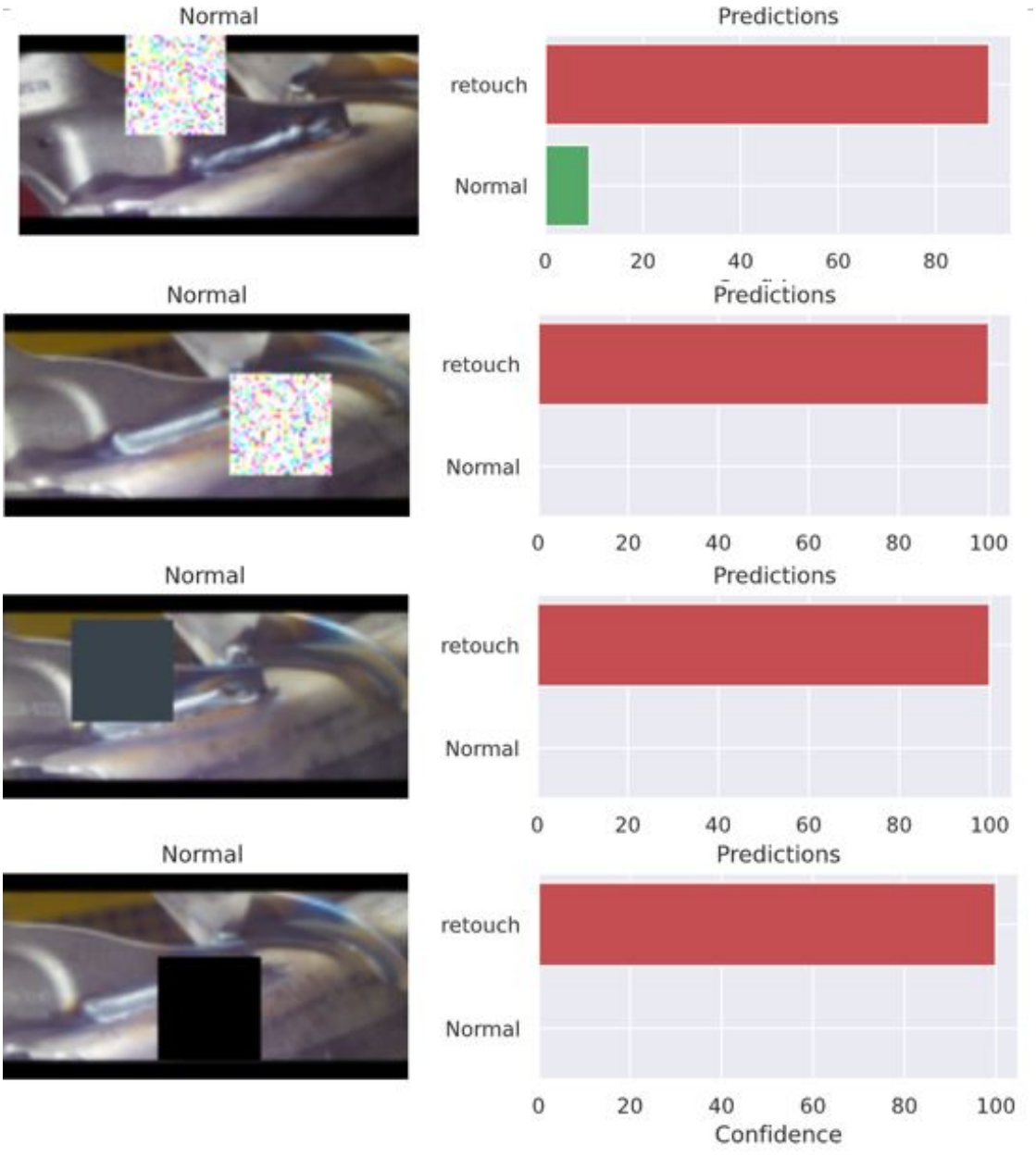
The objective of this section is to evaluate the robustness, against patch attacks, of classification models developed for welding use case. In order to assess the effect of the patch size on the attacks success rate against the models, three sizes were considered: 8×8, 16×16, and 32×32 pixels. Also, in order to check if the degraded performance of the model is not merely caused by the presence of a perturbation, whatever its type is, we considered non optimized patches, simple stickers, generated randomly or simply one-color black or grey squares. The results are given in Table B.4.

As we can notice on table B.4, even with a simple non optimized patch, the model performance drops with 60% or even 72% when using a 32×32 [<- proposition] patch on data from class "Retouch". The grey and black patches are not that successful on data from class "Normal" but the random patches reach 35% misclassification on these data. The optimized patch does, with no surprise, much better than that (62% success rate) with 32x32 size but astonishingly less good results with the smaller patches 16x16 or 8x8. It is an interesting results that must be investigated more thoroughly in the future.

In order to get an idea about the different patches placement on the images and their effect on the classification, some samples are given in Figure B.12 while both classes, "Normal" and "Retouch", are considered in evaluation.

ML Patch	**Patch size 32x32**	**Patch size 16x16**	**Patch size 8x8**
Normal	62.90%	15.58%	7.34%
Retouch	92.89%	27.23%	8.02%
Black patch	**Patch size 32x32**	**Patch size 16x16**	**Patch size 8x8**
Normal	6.45%	9.62%	4.96%
Retouch	72.79%	31.83%	8.55%
Grey patch	**Patch size 32x32**	**Patch size 16x16**	**Patch size 8x8**
Normal	8.23%	10.62%	4.66%
Retouch	67.13%	22.49%	5.56%
Rdm patch	**Patch size 32x32**	**Patch size 16x16**	**Patch size 8x8**
Normal	34.62%	12.10%	8.04%
Retouch	59.85%	64.14%	49.70%
Rdm patch	**Patch size 32x32**	**Patch size 16x16**	**Patch size 8x8**
Normal	33.63%	12.70%	6.75%
Retouch	60.58%	65.10%	50.38%

Table B.4: Patch attacks (first row) and simple perturbations (following four rows) effect on welding UC ML model performance.



B.3.2.1 Conclusion and outlooks

It has been shown in the previous results that the welding classification model is vulnerable, even against very simple patches that reach success rate of more than 70%. This is obviously an alarming situation that must be taken into account in the future models that must be robustified. Nonetheless, the carried out experiments on patch attacks against welding use case were performed in a purely digital way, by changing directly the images pixels, without testing the effect in a physical context. It would indeed be more realistic (and easier) to change the scene since accessing the system software is much more challenging. The evaluation of the criticality of the situation would be more objective with such a scenario. Finally, the patch attacks were evaluated on the welding use case, a model dedicated to a classification task. It would be interesting to consider the same type of attacks against detection models.

B.4. ML Watermarking

The objective is to investigate how ML watermarking can be used to protect the ownership rights of models creators and ensure traceability of ML models. First, state-of-the-art ML watermarking techniques are implemented and tested. Second, ML watermarking is integrated within the Renault's welding inspection use case and its impact on the accuracy of the concerned model is evaluated.

B.4.1 Brief state of the art

Training deep neural networks is not only computationally expensive but also requires specialist knowledge and vast amounts of training data. Therefore, selling pre-trained models became a new lucrative business model. Unfortunately, once the models are sold, they can be copied and redistributed. Even exposing model as a service does not protect them against copying, as model extraction attacks have shown the ability to reproduce a model hidden behind a MLaaS interface [Tramèr et al., 2016]. Therefore, a novel research track aims at providing traceability tools and techniques that make possible to identify models theft or misuse. Among others solutions, ML watermarking seems particularly promising in the context of protection of ownership rights [Kapusta et al., 2020]. Vaguely inspired by watermarking in the multimedia domain, ML watermarking enables models traceability thanks to the embedding of specially crafted perturbations into the marked models. An ML watermark can be defined as any specific modification of the network that can be used to identify its origins [Uchida et al., 2017, Zhang et al., 2018, Adi et al., 2018].

B.4.1.1 Watermarking techniques

ML watermarking techniques can be roughly divided into two main categories - white-box and black-box - with regards on their verification settings. White-box watermarking techniques consist in embedding of a secret mark (ex. a binary vector) into the models parameters or architecture, therefore allowing the ownership identification only in a white-box setting [Uchida et al., 2017] whereas black-box watermarking enables verification with only access to model's inputs

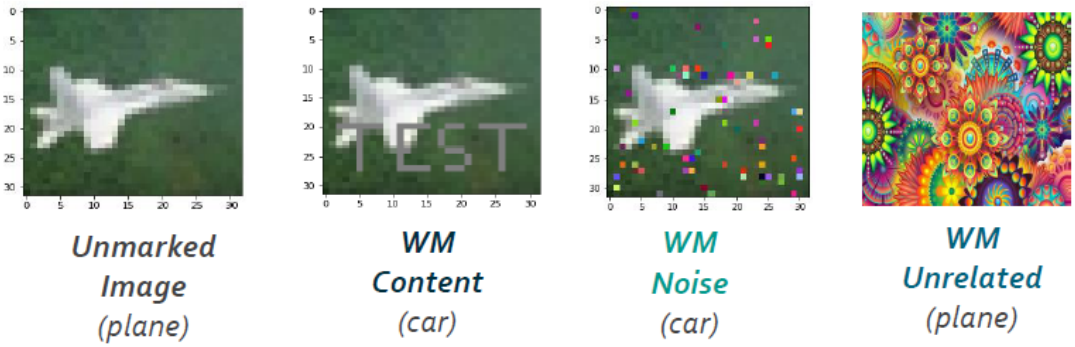


Figure B.13: Illustration of the three main of black-box watermarking present in the literature. From left to right: clean sample from the dataset, **content** watermark, **noise** watermark, and **unrelated** watermark.

and outputs, as the secret mark has the form of an abnormal behavior of the model. White-box watermarking is less practical as it requires access to the model parameters and architecture. The black-box verification is less restrictive and can be performed with only access to the model's API, as it only needs an analysis of reactions of the model to a special set of *key* inputs [Adi et al., 2018, Zhang et al., 2018]. The possibility of verification of the watermark without accessing the model's internals is the major advantage of the black-box approach over the white-box technique. As of now, black-box watermarking techniques are dominating the state-of-the-art.

In more detail, black-box watermarking uses the over-parametrization of the neural networks to modify their behavior for a set of chosen key inputs. In fact, the technique is nothing else than inserting legitimate backdoors into a network by the model creator. There are three main approaches to the generation of the (key input, label) pairs that will be used as watermarks (illustrated in Figure B.13). The first strategy adds a meaningful content to the image of a dataset and changes its label. The second strategy adds irrelevant images to the training set and labels them with a predetermined class. The last strategy injects noise into the images of the datasets and changes their labels.

An efficient black-box watermarking should embed pairs that are clearly tied to the owner's identity and in a way that makes them hard to remove, while preserving the network functionality. During the verification process, the key inputs will be necessarily revealed. Therefore, it is important to embed multiple key inputs in order to allow multiple verification checks.

B.4.1.2 Attacks on watermarking

There are three main categories of attacks against watermarking [Kapusta et al., 2020, Wang and Kerschbaum, 2019]:

- watermark removal, where an attacker tries to remove the watermark from the model. They can use techniques such as fine-tuning, compression or fine-pruning, which will weaken the strength of the marks. Techniques used for backdoor detection and removal

can also serve to erase black-box content watermarks. Moreover, any transformation (intentional or not) of the network may potentially have an impact on the watermarks.

- ambiguity attack, where an attacker will cast doubt on the legitimate ownership by providing counterfeit watermarks. The easiest way to create confusion is to insert a different set of watermarks into an already watermarked model.

- evasion attack, where an attacker will try to escape watermarks verification and therefore disable the model identification. This can be achieved using a query detector that inspects if a query is a clean one or a possible verification attempt.

B.4.2 Preliminary results

The three black-box watermarking techniques were implemented and tested. Preliminary tests were realized on the CIFAR-10 dataset. As watermarking content and noise are similar (watermarking noise can be seen as a variant of the content technique), the focus was put on the comparison between the baseline model and models watermarked with content or unrelated techniques.

Results presented in the Table B.5 have shown that the watermarks are clearly detectable after being embedded during training. The accuracy on the trigger sets for both content and unrelated techniques reaches 100%. When testing the trigger sets on a non-marked model or on a model marked with a different technique, the accuracy drops significantly.

The impact of watermarking on the model accuracy is in Figure B.14. Although at the beginning of the training there are some differences between the baseline and the three marked models, the final accuracy does not significantly differ. The slightly better accuracy of the model marked with the content technique can be at first surprising, but it comes from the fact that initial watermarks where included in the test dataset (and the model learns easily to recognize them without mistake).

Table B.5: Watermarking accuracy for the content and unrelated watermarking approaches.

Model	**Dataset**	**Trigger set content**	**Trigger set unrelated**
model_baseline	91.21%	4%	8%
model_content_100	91.57%	100%	12%
model_unrelated_100	89.2%	11%	93%

A different test aimed at investigating the noise watermarking technique. In more detail, the amount of the generated noise used to mark the key inputs of the watermarks was varied and the accuracy of the watermark detection was measured (see illustration in Figure B.15 and comparison in Table B.6). It shows that clearly detectable watermarking can be achieved with 17% of noise perturbation introduced in the key inputs.

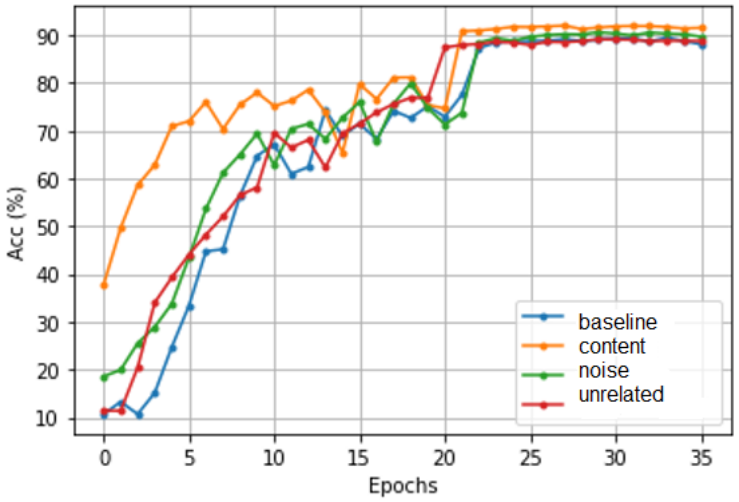


Figure B.14: Comparison of the three main watermarking approaches during training.

Table B.6: Watermarking accuracy for the noise watermarking technique in function of the noise strength (measured as the number of pixels changed in a 32x32 image).

Number of pixels changed	**100px**	**170px**	**256px**	**500px**	**1000px**
Accuracy	44%	96%	100%	100%	100%

B.4.3 Results on the Welding Inspection use case

We implemented and tested the three same black-box watermarking techniques on the Renault Welding inspection use case. For each watermarking technique that we implemented, we compared it to a baseline model with the same architecture: a resnet model composed of 4 hidden layers that we will refer to as "baseline_model" in this document. The training procedure consist of 50 epochs with a SGD optimizer (Stochastic Gradient Descent with momentum) with a starting learning rate of 0.001. Each other model presented in this document will follow the exact same architecture, with the same training procedure — the only differences will be in the presence of various trigger-set during training. When a model is modified, for example during fine-tuning removal attack, we will explain the modifying procedure at that point.

B.4.3.1 Unrelated

The first method we tested on this use case was the **unrelated** watermarking method. It consists in backdooring some data, unrelated to the original dataset during training. For a classification problem, the idea is to assign a specific class to each data point, and train the model with the original dataset of the use case combined with this smaller dataset of unrelated data. This smaller dataset is called the **trigger-set**. Then, during test time, each data point of the trigger-set will be evaluated. In the perfect case, the model should be able to correctly identified each data point of the trigger-set correctly (i.e. to its corresponding class).

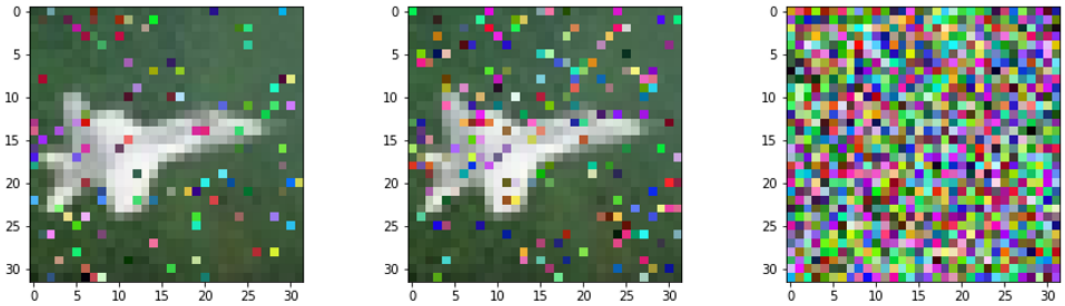


Figure B.15: Varying the amount of noise in the watermarking noise approach by changing pixels to random values. From left to right: 100 pixels changed, 170 pixels changed, 1000 pixels changed .

Table B.7: Nominal result of un-watermarked model (baseline)

Network	**Metrics**			
	Accuracy	Precision	Recall	F1 score
model_baseline	88.08	86.82	99.12	92.56

The strength of the proof of ownership depends on the statistical probability that a random model (i.e. not specifically trained on the trigger-set) would correctly classify each data point of the trigger-set correctly. For a classification problem with M different classes and for a trigger-set of N elements: the probability of a random model to correctly identify all N data points is $\frac{1}{N^M}$. Thus the strength of the ownership proof depends both on the number of classes of the selected use case and the size of the trigger-set. On this specific use case we deal with the smallest possible number of classes for a classification problem as it's a binary classification problem.

For this method we decided to test various sizes of the trigger-set in order to try and increase the watermarking **strength** and proof **capacity** of our model, and to compare the nominal performances of our model to the original use case. The proof capacity corresponds to the number of simultaneous watermarks added to a single model. As a matter of fact: each watermark can only be used once because using it as a proof of ownership would require to expose it. It would then become trivial to removed it.

We hypothesized that the nominal performances should slightly decrease as we increase the size of the trigger set. For that, we used pictures from an open-source dataset: the Stanford-AI Cars dataset [Krause et al., 2013]. It is composed of 16 000 labeled cars images. We created two different trigger-sets, one with only 10 car images and one with 100 images to evaluate the

Figure B.16: Logo used for content watermark.

Table B.8: Comparison of nominal performances for content watermark

Network	**Metrics**			
	Accuracy	Precision	Recall	F1 score
model_baseline	88.08	86.82	99.12	92.56
model_content_10	88.74	86.92	100	93.00
model_content_100	89.40	88.80	98.23	93.28

impact of the size of the trigger-set during the watermarking of the model.

In our first exploration tests, we were able to retrieve the watermark completely for the trigger-set of size 10, without impacting the model's performances. However, for the trigger-set of size 100, we were never able to completely recover the watermark (the best model had a 76% accuracy for the watermark recognition). We should investigate this approach further as it isn't clear whether it is a hard limitation.

B.4.3.2 Content

The second method we decided to test was the **content** watermarking method. It consists in adding a semantic information into a sample of the original dataset, and to train the model into recognizing the semantic information and to classify it in a specific class. In our work we choose the logo depicted in Figure B.16 as our semantic information. Other studies have also used text written directly on the images. We choose to always add it on the top left corner in order to be consistent with its placement, to help the network recognize the watermark. Again, we tested two sizes for the trigger-set: one with 10 images and another with 100 images. For this watermarking method we were able to incorporate the watermark without impacting the model's performances — compared to the "baseline_model" in Table B.8. Further work could focus on trying to add multiple content watermarks to a single model in order to evaluate the proof capacity of such approach to an industrial use case.

B.4.3.3 Noise

The third and last method we tested on this use case was the **noise** watermarking method. Similarly to the two previous methods, it consists in adding a specific information to a sample of

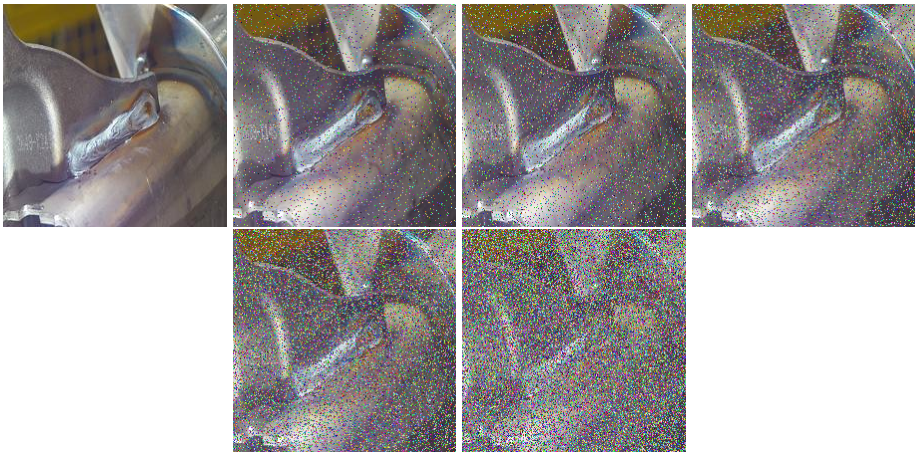


Figure B.17: Various amount of noise added for the noise watermark

the dataset, to try to make the model learn the association between this information and a specific class. Opposite to the content method, here, the added information does not contain any meaningful semantic information. It consists of a specific random noise. More precisely, for a data recognition problem, a number of pixels will be changed to correspond to a specific random pattern. For each image of the trigger-set the same noise is applied.

For the noise watermarking method, we tested various amount of noise to add in each pictures of the trigger-set. We expected that, the more noise was added, the more identifiable the watermark would become. On the other hand, adding too much noise could decrease the performances of the model. Moreover, if the noise becomes too important and covers the majority of each picture of the trigger-set, then, each data point of the trigger-set will became approximately the same. It would greatly reduce the strength of the ownership proof. In order to avoid that, the images used for the trigger-set should not entirely be transformed into noise.

We chose the amount of noise according to previous studies summarized in Table B.6 and decided to evaluate the watermark recognition accuracy of the same absolute total information of noise and the same relative information of noise. As the previous study was done on 32×32 images and we are evaluating it on 224×224 images we evaluated the watermark for the following number of modified pixels: {100, 170, 256, 500, 1 000, 4 900, 8 330, 12 540, 24 500, 49 000}.

We were able to reproduce the previous results obtained on the public dataset of 32×32 images for the same relative amount of modified data used for the noise watermark. Thus confirming that a minimal quantity of noise is necessary to be able to recognize the watermark. We also evaluated the influence of the size of the trigger-set over the watermark recognition as we hypothesized that the size of the trigger-set could have an influence over the recognition of the watermark. Figure B.18 shows that a larger trigger-set can result in a greater recognition of the watermark for some amount of noise included (here: 4 900 and 8 330 pixels changed).

On the other hand, we also analyzed the performances of the model on the use case to see if

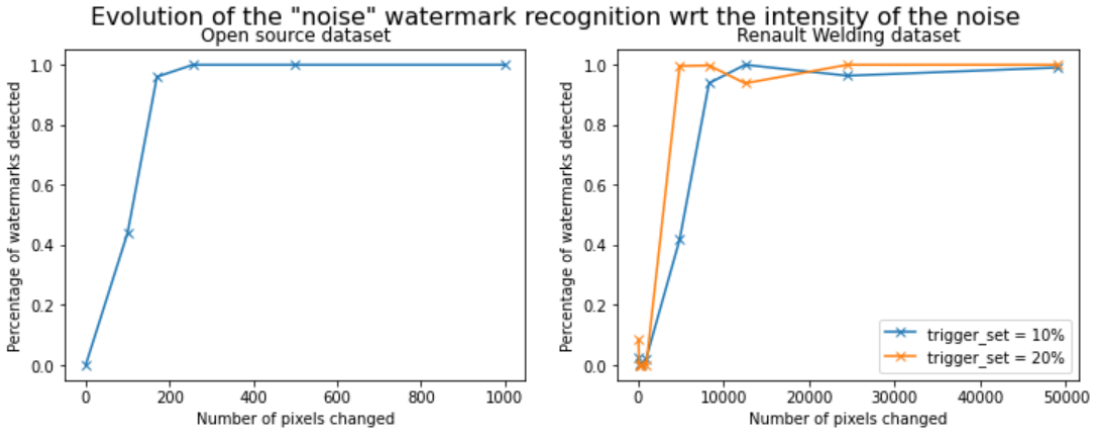


Figure B.18: Comparison of noise watermark recognition accuracy based on the number of modified pixels

the addition of the noise watermark was any detrimental to the model's nominal performances. We observed that for a small noise the training performances of the model were completely reduced. For each watermark added with less than 4 900 pixels modified, the networks were not able to satisfyingly classify the welding pictures (see Figure B.19).

We interpret this result as the fact that the watermark creates a competition for the classification of originally correct weld pictures (classified as "OK"). Thus, when we only add a small noise, it is more difficult for the model to differentiate between noisy and non-noisy images (watermarked and un-watermarked images). On the other hand, when the noise is more important, the model can more easily learn the noise pattern rather than the semantic information presented in the weld pictures used for the trigger-set.

B.4.3.4 Fine-tuning attack

Another scope of this study was to evaluate the robustness of these watermarks. For this reason, we evaluated the fine-tuning attack for watermark removal (see Section B.4.1.2 on the attacks on watermarking). We evaluated various number of fine-tuning epochs over a model already-trained with a watermark. As preliminary results were unable to remove the watermark, we tried a more aggressive fine-tuning attack. We chose a learning rate of 10^{-5} for 10, 15 and 20 epochs. With this procedure we where not able to remove the watermark as all the test data used for watermark recognition were still correctly classified by the model, even though we were reducing the model nominal performances significantly (see Table B.20).

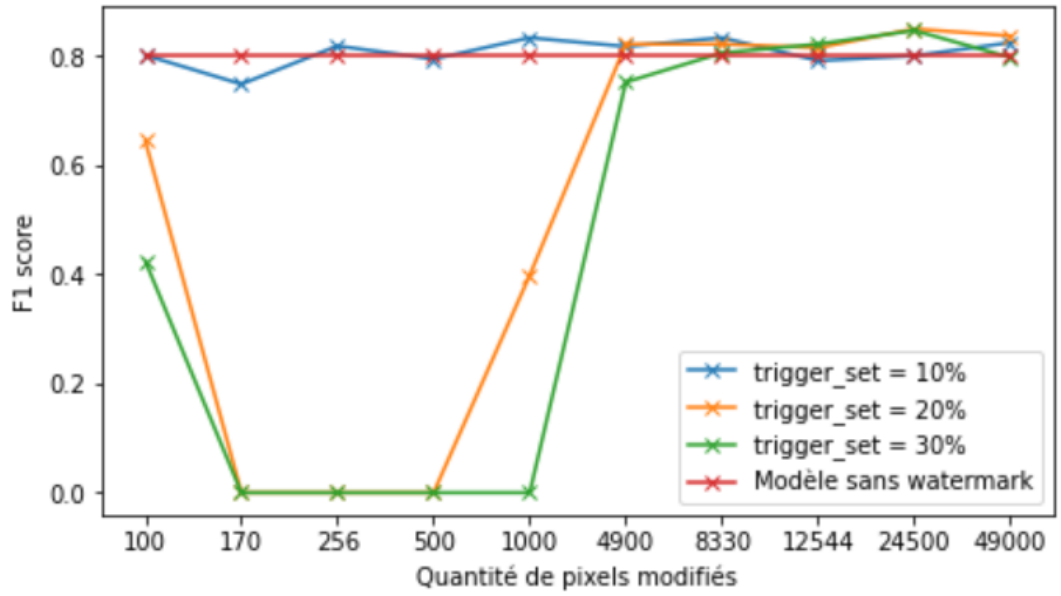


Figure B.19: Comparison of nominal performance for noise watermark based on the number of modified pixels

	Model performances		Watermark recognition
	Accuracy	F1_score	Watermark recognition
Before fine tunning atack	93.23%	85.02%	**100.00%**
After 10 epochs	92.57%	82.82%	**100.00%**
After 15 epochs	91.05%	78.53%	**100.00%**
After 20 epochs	89.96%	74.99%	**100.00%**

Figure B.20: Evolution of model's performances during the fine-tuning attack

B.5. Visualization Tools

B.5.1 Introduction

This section aims to present visualization tools that can be used to represent and analyze data generated from the different Environment Alteration frameworks. First, we introduce the notion of Data Visualization. Then we present DebiAI (a visualization tool developed by IRT-SystemX), its workflow, a practical example and finally some benefits and limitations.

B.5.2 Data Visualization

Data visualization is a field that deals with the graphic representation of data and information. It is a particularly efficient way of communicating when the data is numerous, for example a time series.

This representation can be considered as a mapping between the original data and graphic elements (lines or points in a chart). The mapping determines how the attributes of these elements vary according to the data. In this light, a bar chart is a mapping of the length of a bar to a magnitude of a variable.

In other words, data visualization is the practice of translating complex data into a visual context, such as a map or graph, to make data easier to understand and pull insights from. The main goal of data visualization is to make it easier to identify patterns, trends, and outliers in large data sets.

B.5.3 DebiAI

DebiAI is an open-source web data exploration and visualization application that aims to facilitate the process of developing Machine Learning models, especially in the stage of: data analysis and the model performance comparison. DebiAI provides features to:

1. Identify biases and errors in the input, results, contextual[4] or ground truth data;
2. Make a comparison of the performance of ML models according to their contextual results;
3. Select and create sets of data graphically for further analysis or (re-)training purposes;
4. Quickly create and share statistical visualizations of data.

DebiAI has a Graphical User Interface with a complete data visualization toolkit offering many statistical analysis tools.

B.5.3.1 Workflow

DebiAI is a high level data exploration tool for data scientists and machine learning experts. It is designed to be easily integrated in project workflow.

[4] a context is all attributes from data that provide meaning to the dataset's background

The main way to provide data to the application is through the DebiAI Python module. The module was designed to be used directly in the Python workflow, to add model results directly after its evaluation for example.

DEBIAI can be used both before the modeling phase, to understand, characterize and summarize data, and afterwards, to interpret the result of a model with respect to input data, contexts or features.

The workflow can be decomposed into:

1. Project creation
    - Data formatting
    - Data loading
2. Model evaluation
    - Results insertion
    - Results contextual comparison
3. Data analysis
    - Data visualization
    - Data selection

B.5.3.2 Example: Randomized smoothing applied to Renault UC

Project creation

```
DEBIAI_BACKEND_URL = "http://localhost:3000/"
DEBIAI_PROJECT_NAME = "Renault Welding UC"

# Initialisation
my_debiai = debiai.Debiai(DEBIAI_BACKEND_URL)

# Creating a project
debiai_project = my_debiai.create_project(DEBIAI_PROJECT_NAME)
```

Figure B.21: Project creation code

First, we have to run the DebiAI instance and configure a specific address by specifying `DEBIAI_BACKEND_URL`. Then create a project by specifying its name `(DEBIAI_PROJECT_NAME)`, as seen in Figure B.21.

(ie: If the project already exists, you can get the project with:
debiai_project = my_debiai.get_project(DEBIAI_PROJECT_NAME))

Project data insertion

Each data point that you want to insert must be associated with:

- An **ID**: considering that the dataset and the results are inserted in two different steps, an ID is recquired for each inserted data in order to match them later.
- A **type**: `text`, `number` or `boolean`.
- A **label**: `inputs`, `groundTruth`, `contexts`, or others.

The type and the label are used for authorizing (or suggesting) specific data manipulation or visualizations.

To do so, a block structure must be defined, with at least one object containing the following keys see Figure B.22:

- `name`: for setting the ID column
- `inputs`, `groundTruth`, `contexts`, or others: optional lists with the type and the name of the columns of your dataset.

The added data need to follow the previously defined block structure. They can be numpy array or pandas DataFrame as in Figure B.23.

```
block_structure = [
    {
        "name": "Data ID",
        "contexts": [
            {"name": 'Name', "type": "text"} ,
        ],
        "groundTruth": [
            {"name": 'Approved', "type": "number"}
        ]
    }
]

debiai_project.set_blockstructure(block_structure)
```

Figure B.22: Data block structure

```
samples_df = pd.DataFrame({
    "Data ID":          ["Data-1", "Data-2", "Data-3"],
    "Name":             ["A", "B", "C"],
    "Approved":         [1, 1, 0]
})
```

Figure B.23: Data Sample

Model metadata and model results insertion

The first step is to set, for the results, the equivalent of the block structure: each result must have an **ID** (to map them with the data), and a **type**.

To do so, an **expected results** must be defined (the equivalent of block structure, but for the results), with a list of objects containing the "name" of the column, and the "type" ("text", "number" or "boolean"). There is no need to set the ID column: the ID column is set by default, with the same name it has in the block structureB.24.

The added data need to follow the previously defined block structure.

```
expected_results = [
    {"name": "Model result",    "type": "number"},
    {"name": "Model name",      "type": "text"},
    {"name": "Attack name",     "type": "text"},
    {"name": "Attack power",    "type": "number"}
]

debiai_project.set_expected_results(expected_results)
```

Figure B.24: Result block structure

Data Visualization

In this section we will illustrate the use of DebiAi on a use case. The chosen use case is Renault's welding inspection. We will compare the different results obtained on applying adversarial attacks (FGSM and PGD) and robustification techniques (randomized smoothing). For more information or mathematical details, consult the deliverable **ISX-EC4-LIV-1498** of **batch 1** entitled "Robust & Embeddable Deep Learning by Design".

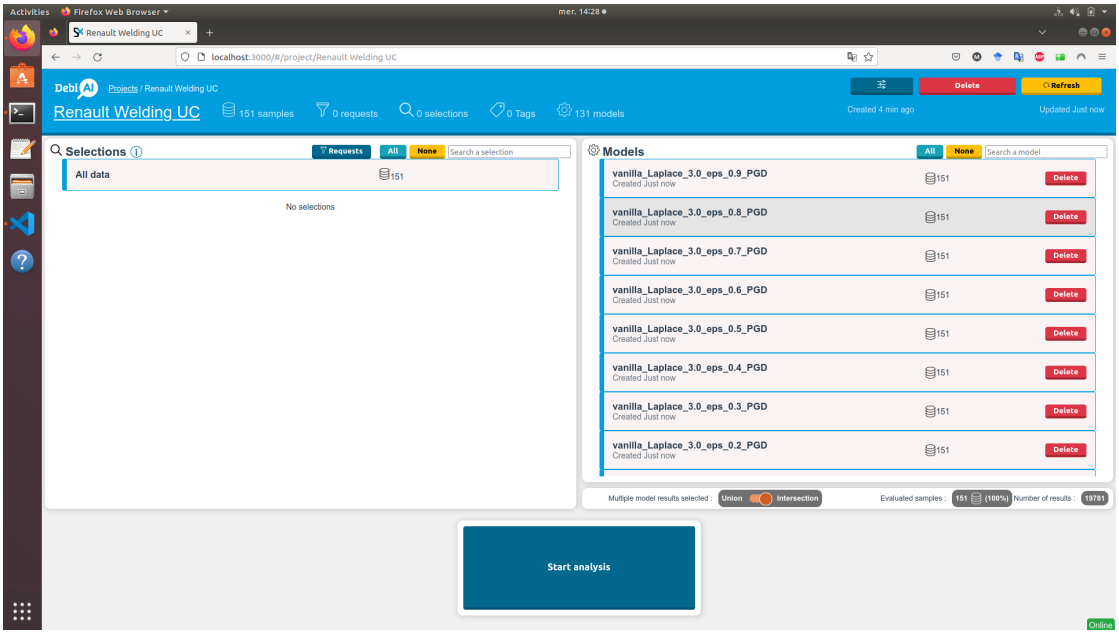


Figure B.25: Data Selection Phase

Dashboard: Data Selection

Widget: Point plot

The point plot displays points in 2D space and averages as line chart, bar chart or scatter plot. It can be used to compare models' performances with the project contexts.

Line Chart - Average Results: The figure B.26 shows a visualization of the average precision of 7 attacked models (1 vanilla model and 6 models trained by randomized smoothing).

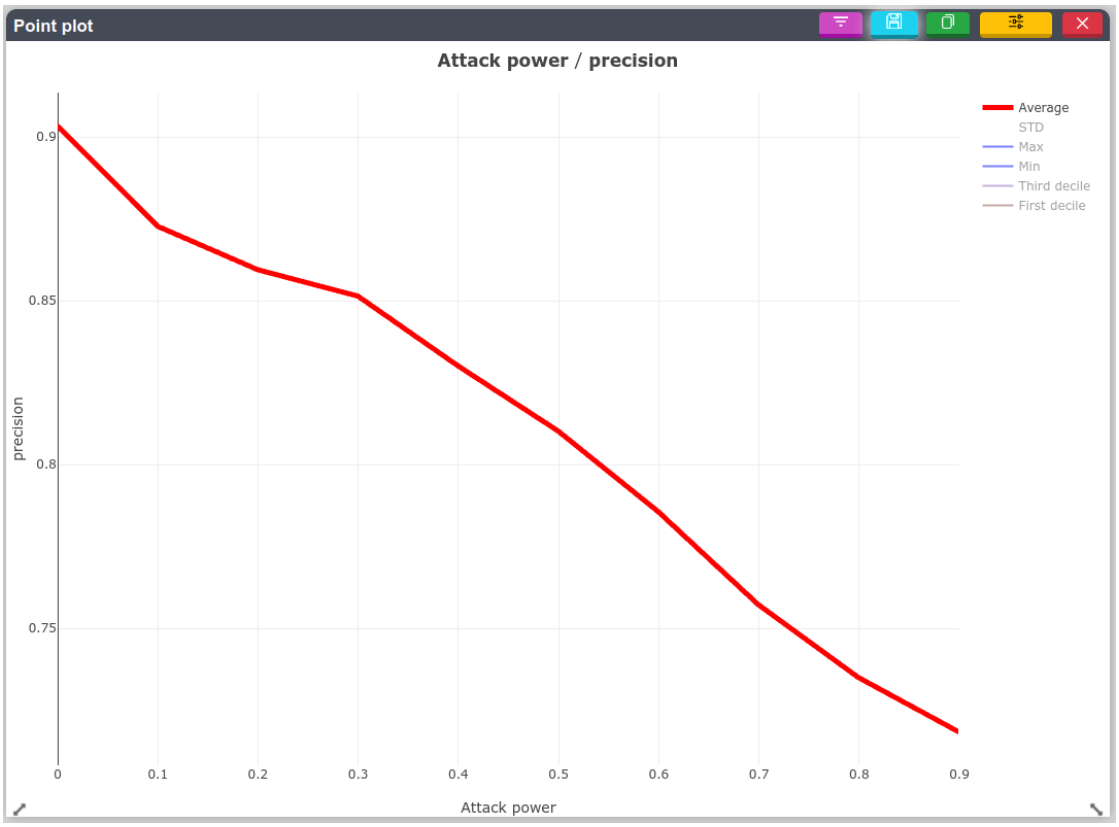


Figure B.26: Line Chart: Average Results

Line Chart - Results Grouped by "Tags": Figures B.27 and B.28 show one interesting feature of DebiAI: grouping results according to a "tag" (ex. "model name", "Attack name", ...)

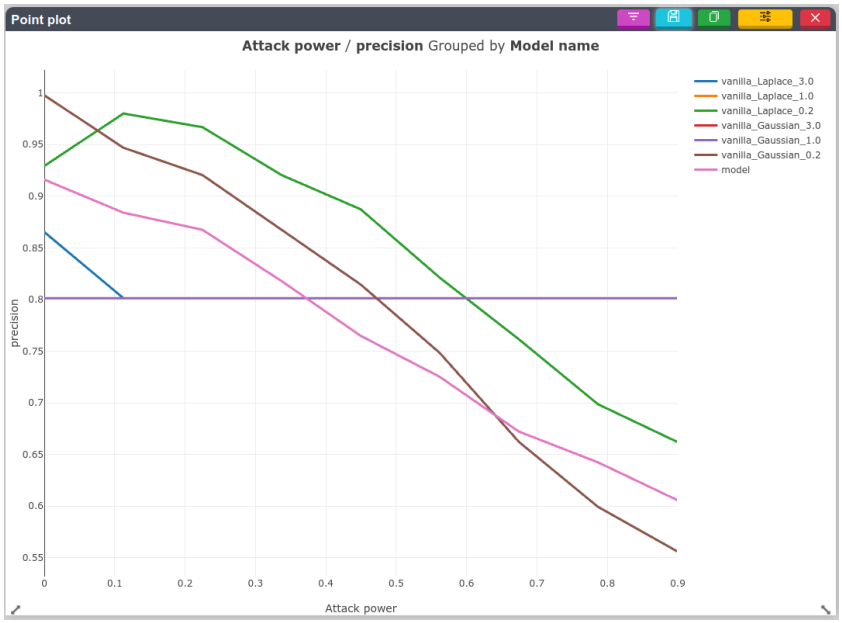


Figure B.27: Line Chart: Results Grouped by Network Name

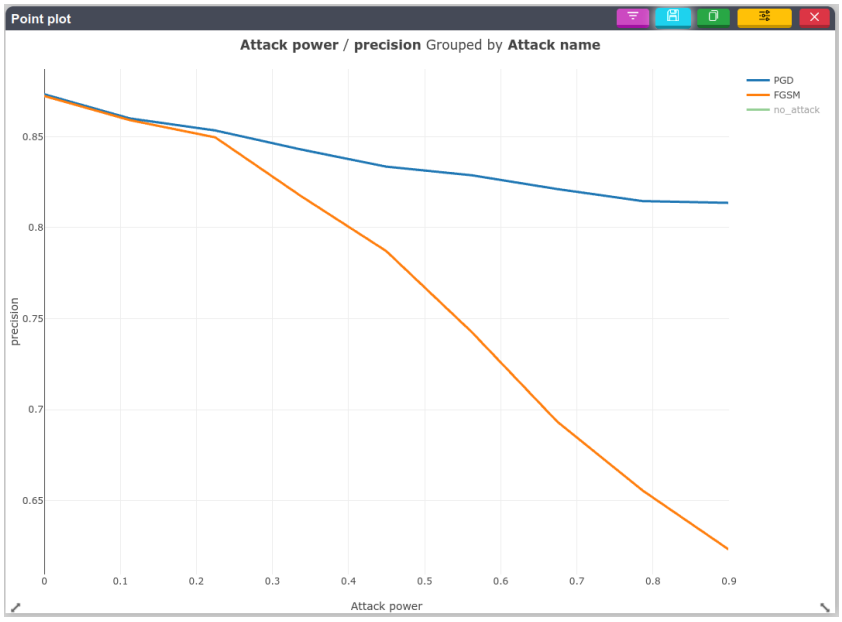


Figure B.28: Line Chart: Results Grouped by Attack Name

Widget: Data distribution

The data distribution widget displays the distribution of a given feature or result. It can be used to compare models' inputs and outputs.

Average Results: The figure B.29 shows a visualization of the models' result distribution.

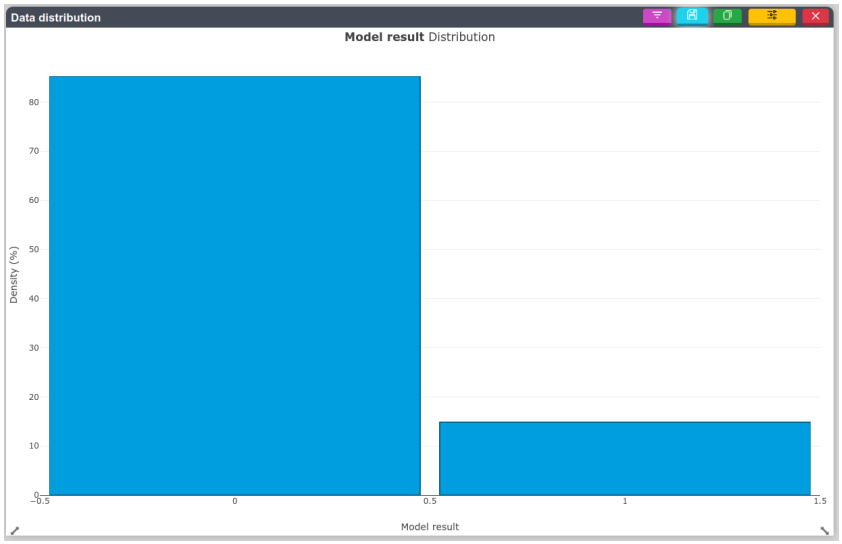


Figure B.29: Distribution Visualization: Raw Results

Results Grouped by Attack Name: The figure B.30 shows a visualization of the models' result distribution grouped by the type of attack.

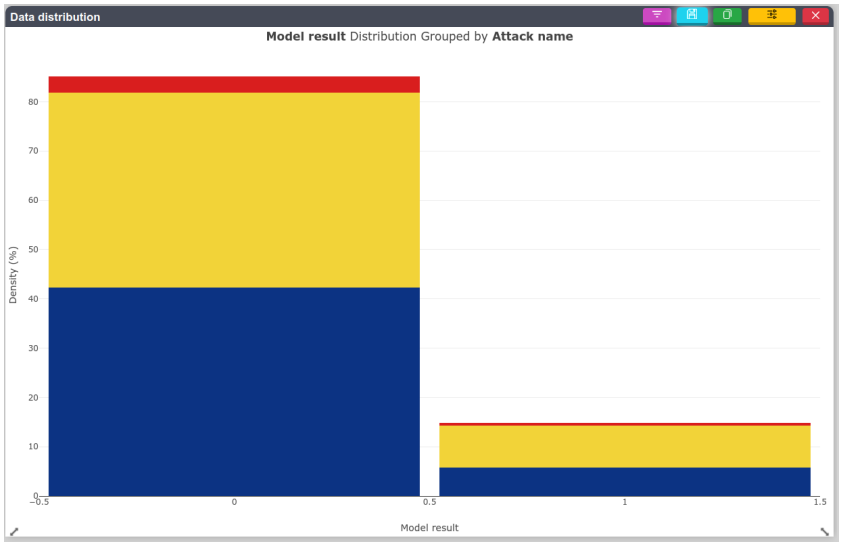


Figure B.30: Distribution Visualization: Results Grouped by Attack Name

Widget: Confusion Matrix

The confusion matrix widget is useful for classification problems. It shows the number of true positives, true negatives, false positives, and false negativesB.31.

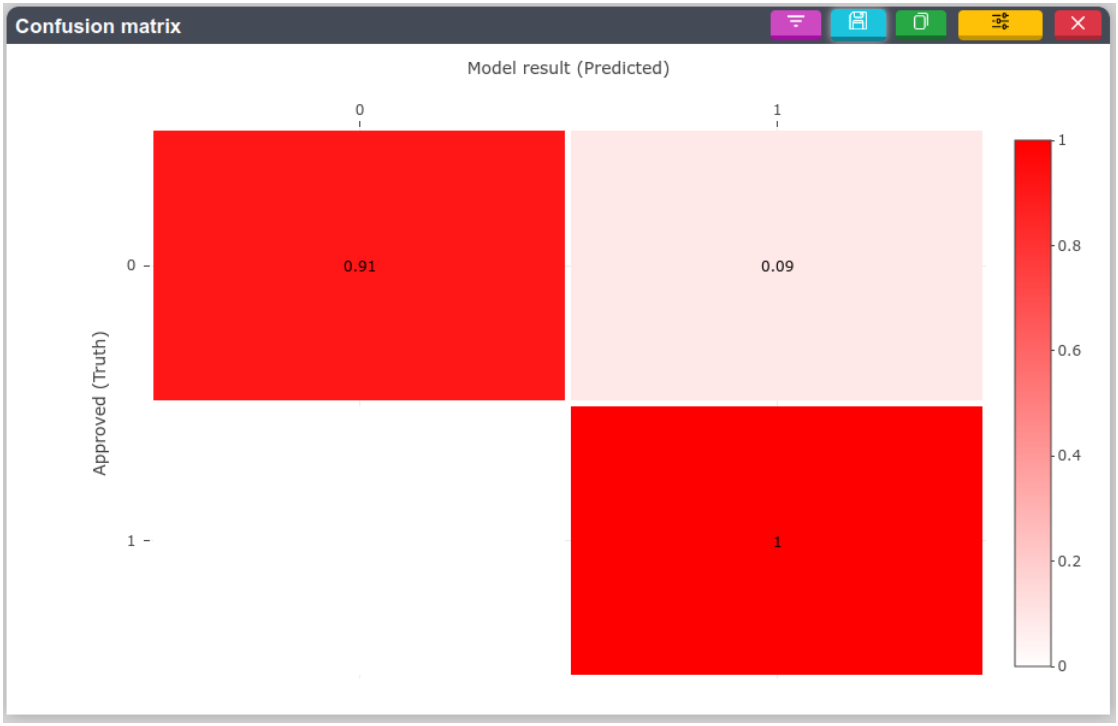


Figure B.31: Confusion Matrix: Average Results

Widget: Statistics

The statistics widget displays the average, standard deviation, and minimum and maximum values of the selected data points for a feature B.32.

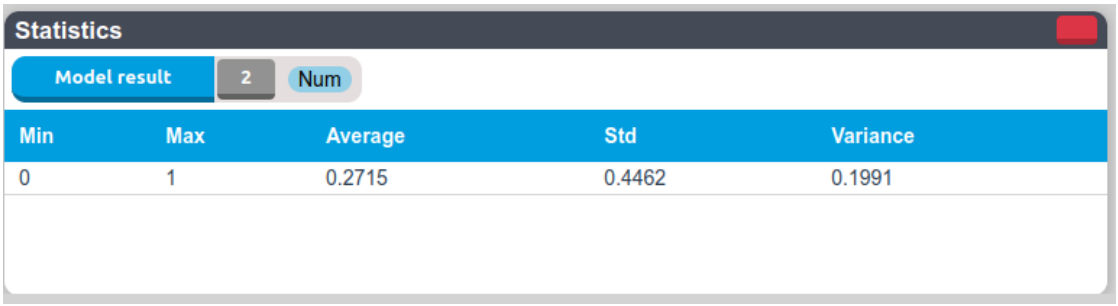


Figure B.32: Standard Statistics

B.5.4 Conclusion

The benefits of using DebiAI

- It can be used with many use cases (images, time series ...)
- It helps find bias in the data
- It gives a lot of meaning to the contextual data
- It is well-suited for live presentations

Current limitations

- DEBIAI is in beta. A release is already available and has given very interesting results on many AI projects (ex. EC1, EC4, EC5, ...).
- There is a maximum sample count: a DebiAI project with more than 100.000 samples will have some performance issues when using specific visualization widgets. It can be noted that some of the projects that have been using DebiAI have been able to make some successful analysis with more than 350.000 samples.
- Display of images or signals: In its current version, DEBIAI does not allow the display of input data such as images or signals, but it does allow the display and analysis of data related to these images or signals, which are then considered contextual data.

B.6. Conclusion

In this chapter, we examined the behavior of AI models designed to be robust against environmental alterations such as evasion attacks and adversarial patches. The obtained results demonstrated that models trained by adversarial training are more robust than models trained by randomized smoothing. We were able to show the vulnerability of industrial models to patch-attacks, even for not optimized patches. For future works we would like to be able to add physical patches to the Renault's Welding use case rather than create digital patches to the dataset. In addition, we evaluated three state-of-the-art ML watermarking techniques (content, noise, unrelated) applied to the Renault's Welding Inspection use case. The obtained results demonstrated that all three ML watermarks introduced into the marked model are well detectable and thus allow to identify the network. Moreover, we tested the robustness of the noise-based watermarking against removal attempts based on fine-tuning. First results have shown that watermarking can resist fine-tuning. On this topic further works should be produced to fully evaluate watermarking techniques and their robustness. Lastly, we presented DebiAI, a web data visualization application. We tested DebiAI on a use case and have shown its usefulness in result analysis and comparison, as its ease of use and flexibility.

Chapter C

Robustification methods based on Neural Differential Equations

Contents

We investigate the problems and challenges of evaluating the robustness of Differential Equation-based (DE) networks against synthetic distribution shifts. We propose a novel and simple accuracy metric which can be used to evaluate intrinsic robustness and to validate dataset corruption simulators. We also propose methodology recommendations for evaluating different aspects of neural DEs' robustness and comparing them with their discrete counterparts rigorously. We then use this criteria to evaluate an inexpensive data augmentation technique as a reliable way for demonstrating the natural robustness of neural ODEs against simulated image corruptions across multiple datasets. Finally, we provide a solid image purification benchmark with Air Liquide's cylinder counting dataset, a novel corruption-robust system architecture and in depth validation.

C.1. Introduction

Neural Ordinary Differential Equations (NODEs) Chen et al. [2019b], conjoining dynamical systems (DS) and machine learning (ML), have come to be a popular source of interest, in particular for tackling generative problems and continuous-time modeling, and seem to have a bright future among the ML community. Nevertheless, many questions regarding their robustness have been raised, creating a debate on whether these networks benefit from natural robustness properties or if the latter are overestimated.

Likewise, the importance of *reliability* in real-world applications with AI-driven decision-making in safety-critical systems have brought a lot of attention to studying a model's behavior under *distribution shifts*. Understanding the latter implies focusing on how feasible is a chosen model's domain generalization against the kinds of shifts that may occur in real-world scenarios. The past few years have seen an emerging industry proposing new and relevant shifted datasets for different actors and purposes. Numerous benchmarks Hendrycks and Dietterich [2019], Mu and Gilmer [2019], Koh et al. [2021], Salehi et al. [2021] addressing different aspects of distribution shifts have come to light and the rigorous analysis and evaluation of both models and benchmarks have become increasingly important. Although they transfer poorly to real-world shifted images, synthetic distribution shifts are a good starting point for experimenting a new model's accuracy and robustness. For instance, in Gilmer et al. [2019] it is hypothesized that methods that incur into vanishing gradients also show no improvement in Gaussian noise, a phenomenon which they relate in a rigorous way to adversarial attacks. Corruption robustness can be then seen as a *sanity check* to ensure that a proposed adversarial defense method doesn't present gradient masking. Nevertheless, it is important to separate accuracy improvements from robustness improvements when interpreting the results and different metrics have been proposed for doing so Hendrycks et al. [2021], Taori et al. [2020]. For common corruptions Hendrycks and Dietterich [2019], the (un-normalized, unaveraged) relative Corruption Error (rCE) is the difference[1] of the model's corrupted and clean errors. As the very notion of a corruption is always relative to a clean counterpart, we find that this metric has a particular weakness for simulated corruptions as it doesn't take into account the following structural principle underlying such corruptions: *miss-classified clean images should result in miss-classified simulated corruptions*. As such, the rCE answers questions like "how much does the model decline under corruption inputs" but it doesn't detect the corruption error contributions coming from clean miss-classifications.

In this brief account we lay initial ground on theoretical and application-driven aspects, problems and methodology perspectives for evaluating robustness of NODEs against synthetic distribution shifts. For this purpose, we

1. assess and highlight several properties of NODEs in connection to different robustness criteria, link them to specific aspects of real-world data features they may capture and determine general guidelines on when and how they can be compared to static networks or between them;

2. introduce an intrinsic robustness metric $\mathscr{A}_c^{\text{rel}}$, well-suited for evaluating well-posedness

[1] Precise definitions are recalled in (C.13)

of dataset corruption simulators, and capable of measuring a model's corruption accuracy more subtly than the rCE Hendrycks and Dietterich [2019];

3. evaluate an easy-to-implement robustifying method for NODEs against corrupted images, leading us to conclude that NODEs are naturally more robust to several synthetic distribution shifts than their discrete counterparts and that noisy learning for NODEs acts, as expected, as a robustness locus widening.

We aim to propose a baseline upon which to build step-wise incremental implementations of robustifying methods for NODEs under such corruptions. It is our hope that our proposed metric and methodology recommendations will be helpful both when studying implicit nets robustness and when designing new and more diverse corruption simulation algorithms and datasets.

C.2. On evaluating common robustness for neural DEs

Evaluating robustness of NODEs is particularly challenging: their output is computed via iterative optimization schemes and such test-time optimization has shown to prevent the proper evaluation of established robustness methods designed for static networks like AUTOATTACK in the adversarial context Croce et al. [2022]. Additionally, comparing NODEs to chosen static analogs has shown to disregard implicit assumptions (adaptive step-size solvers, inexact backward pass computation) which pose methodological problems preventing to formally compare them and ultimately incurs into falsifying the results of many conducted experiments. We will concentrate on classification tasks in this report.

Neural DEs meet dynamical systems: Denote h_x a feature extractor (FE) and h_y a fully-connected classifier (FCC). The inference of a NODE model is carried out by solving,for $\dot{z}$ denoting the time-derivative:

$$\begin{cases} \dot{z}(t) = f(t, z(t), \theta(t), x) & \\ z(0) = h_x(x) & t \in \mathscr{T}_x = [0, T_x] \\ \hat{y}(T_x) = h_y(z(T_x)) & \end{cases} \tag{C.1}$$

Contrary to static architectures, f formalizes the dynamics controlling a *continuous-in-depth* model, $t \in \mathscr{T}_x$ being its depth variable and the components in (C.1) traduce the following features: the dependence on h_x for $z(0)$ is traduced by input layer augmentation; the dependence on t for f (resp. θ) is traduced by depth-dependence (resp. depth-variance[2]) and is taken in practice as an augmentation component Dupont et al. [2019]; the dependence on x for f (resp. T_x) is traduced as data-control (resp. depth-adaptation) and can traduce recurrent architectures. We refer to Massaroli et al. [2020] for details on these features and to Kidger [2022] for a clear comprehensive introduction to neural DEs.

In Figure C.1 we rapidly illustrated how neural differential equations work.

- In inference, layers are determined as numerical scheme of a differential equation, so that the inference time is proportional to the number of function evaluations used by such solver times times the number of steps used by the numerical scheme;

[2]When θ is a constant function, we still use the term depth-dependence.

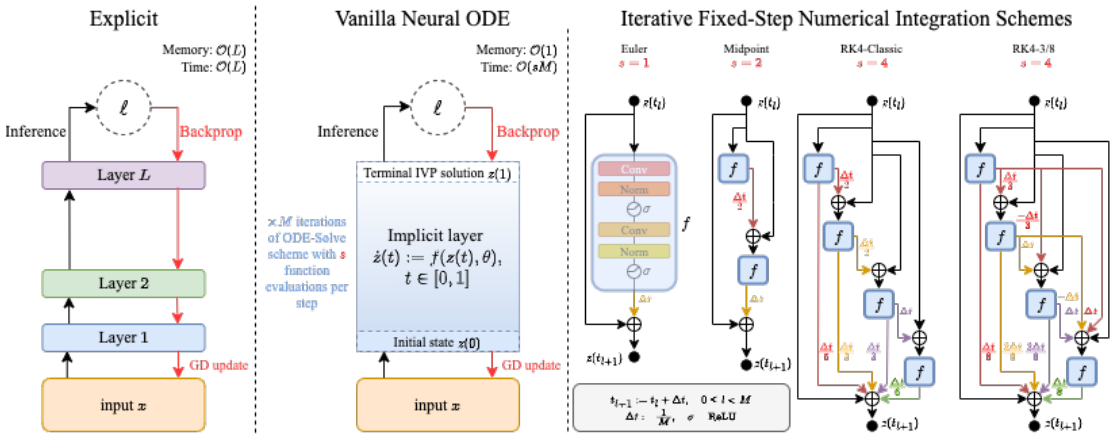
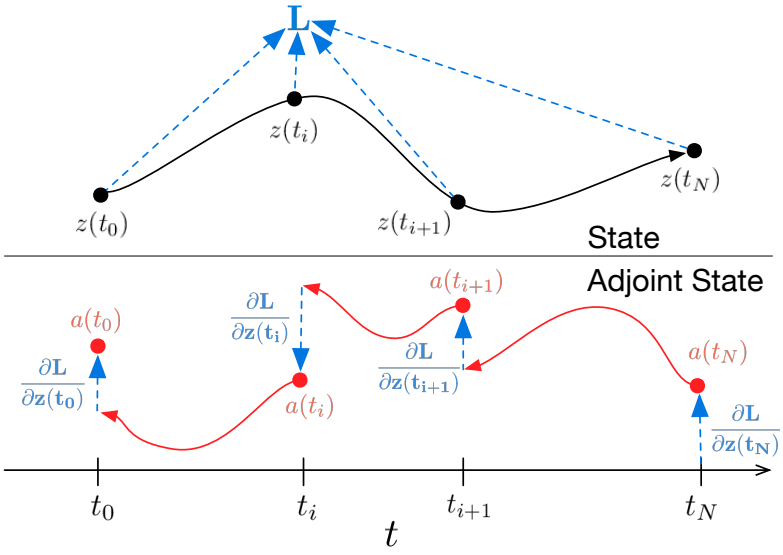


Figure C.1: A Bird's eye view of the mechanism of Neural Differential Equations.

- Gradients can be either stored to be used in the usual way (for back-propagation or for crafting adaptive adversary attacks) which could be very memory expensive or *not stored*. In the last scenario, one uses the so-called adjoint equation of the inference differential equation, going backwards in inference time, and run this equation in parallel to it in order to obtain the gradients.



- If using the *adjoint method*, learning is done with constant memory since we do not store intermediate gradients but there are not *a priori* guarantees that the learned NDE trajectories will be optimal - and thus necessitating a sub-optimal number of inference NFEs.

Needless to say, several assumptions need to be satisfied by the different components in (C.1) in order to meet desired properties (such as non intersecting trajectories as ensured by the Picard-Lindelöf theorem). We refer to Kidger [2022] for technical details on these properties.

Several overlaps occur between ML and DS modeling techniques and approaches which we now try to articulate to shed light on their singular benefits. These distinctions will be the basis of our methodology guidelines for evaluating and comparing general Neural Differential Equation (NDE) models.

C.2.1 DS-inspired neural DEs

These consist in manufacturing constraints on a loss function or on the weight matrices inside the dynamics that would enhance their robustness from a stability analysis point of view. Another way of stating DS-inspired NDEs is to say that the ML focus comes **post-hoc** the DS focus: the trained architecture is supposed to have benefit of theoretical properties at training or inference. We highlight the fact that such approaches can serve different purposes: Pal et al. [2022], Ivan et al. [2022], Djeumou et al. [2022] address mainly speed problems while Kang et al. [2021], Yan et al. [2020], Huang et al. [2022] addresses stability training considerations Li et al. [2019] for NDEs e.g. using steady-states, Lyapunov equilibrium points. This usually is an idealized analysis made upon an idealized ML architecture. A common problem would be to neglect the numerical errors that come to hand while training, which were absent from classical discrete neural networks. First, in Ott et al. [2021] it is shown that there exists a critical step size only beyond which the training yields a valid ODE vector field. Thus, for instance, the system theoretic formulation of the Picard-Lindelöf theorem, used for ensuring non intersecting trajectories, effectively applies only if such condition is met.

Methodology point: ensure that the discretization resulting from the numerical solver's execution preserves formalized DS properties. A first characterization of robustness for NDEs is then met.

C.2.2 DS-based neural DEs

These consist in formalizing NDE architectures as analogs of system theoretic paradigms such as a full use of the components of (C.1) but also formalizing neural CDEs, SDEs and PDEs Kidger [2022], Fermanian et al. [2021], Xu et al. [2022], Li et al. [2021]. Here, the ML focus comes **ante-hoc** the DS focus: the formalized architecture is supposed to be endowed with structural characteristics both at training and inference. As an example, inference in neural controlled DEs will be based on Riemann–Stieltjes integrals of the form

$$\int_0^t f_\theta(y(s))\,\mathrm{d}x(s) = \int_0^t f(y(s))\frac{\mathrm{d}x}{\mathrm{d}s}(s)\,\mathrm{d}s, \tag{C.2}$$

while the inference of neural stochastic DEs will consist of the sum of a deterministic integral and an Ito or Stratonovich stochastic integral:

$$\int_0^t f_\theta(s,y(s))\mathrm{d}s + \int_0^t g_\theta(s,y(s)) \circ \mathrm{d}B(s). \tag{C.3}$$

This approach is more involved than the previous one as it needs to have at hand simultaneously an adapted, analytically proven, adjoint method analog or generalization, and a non empty

choice of adapted numerical solvers. For instance, Kidger Kidger [2022] adapted the analytic adjoint method for neural CDEs and neural SDEs while developing for the latter an algebraically reversible *Heun method* SDE solver. While NODEs have served as inspiration for constructing many discrete neural architectures by formalizing in continuous-time an ODE and discretizing it, the difficulty has been to create continuous time analogs for discrete neural components. For instance, crafting a stateful batch normalization (BN) layer has been recently reflected in the NODE formulation Queiruga et al. [2021] as a generalized ODE. On the contrary, in Huang et al. [2022] ResNets have BN layers, NODEs have group normalization (GN) layers and the length of the skip connection does not coincide between the compared architectures and in Xu et al. [2022], although testing NODEs against corruptions, a mix between deterministic and stochastic methods may weaken their claims.

Methodology point: ensure that the chosen NDE architecture identifies in a clear manner all arguments of the function passed to the DE solver, determine if the latter is an exact or an approximate solver; distinguish stochastic and deterministic architectures; comparing NDEs and discrete architectures should be mathematically justified by an explicit end-to-end discretization scheme, taking into account the nature of the h_x and h_y layers and identifying, for instance, NODE blocks and weight-tied residual blocks. This constitutes a second robustness characterization.

C.2.3 DS-destined neural DEs

These consist in manufacturing NDEs that incorporate known modeling physical constraints. Here, the ML focus is **ad-hoc** to the DS focus: the proposed architecture is supposed to capture *intrinsically* the dynamics (e.g. Lagrangians, Hamiltonians) of the studied phenomenon and NDEs *specify* DEs Zhong et al. [2020]. This is different, though somehow related, to *physics-informed* NNs Karniadakis et al. [2021], which aim to obtain solutions through NNs to *pre-specified* DEs for which traditional solvers are computationally expensive. Well-defined NDEs may not effectively capture continuous-time inductive biases if the used numerical ODE methods have too low order of convergence while high-order methods need for fast and exact gradient computations Matsubara et al. [2021], Djeumou et al. [2022].

Methodology point: conduct NDE-oriented numerical *convergence* tests, presenting a non-trivial difference between the ML-based Bottou and Bousquet [2007] and the round-off Chaitin-Chatelin and Frayssé [1996] numerical errors, such as proposed in Krishnapriyan et al. [2022] to check if the implemented model successfully learned meaningful continuous dynamics. We then find a third robustness characterization for such networks.

Leveraging well-studied mathematical approaches for stability, robustness and resilience[3] into the continuous-time ML community can prove to be very advantageous and the above robustness properties, while already being studied jointly in the cited works, call for clear distinctions between such notions. Their formal analysis will be the subject of an extended version of the present report, complementary to the system-theoretic approach which is being conducted in parallel Gonzalez et al. [2022a].

[3]For instance, NODEs with depth-adaptation should be evaluate their recovery rate in time *after* an input perturbation.

C.2.4 Intrinsic robustness metrics

We now define the metric $\mathscr{A}_c^{\text{rel}}$ mentioned in the introduction of this report. Let C be a set of simulated corruptions $c = (\tilde{c}, s)$, where $\tilde{c}$ is a corruption label and s is a severity level. We make the assumption that for each $c \in C$ one can generate (at least) one corruption simulation x_c of a clean image x. We denote y the true label of x, y_{cl} the model's prediction on x and y_c the model's prediction on x_c. Let N be the size of the dataset. Define $M = \sum_{i=1}^{N} \mathbb{I}_{\{y_{\text{cl}}^i = y^i\}}$, $\mathscr{A}_{\text{cl}} = M/N$, and for $c \in C$,

$$\mathscr{A}_c = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{\{y_c^i = y^i\}}, \quad \mathscr{A}_c^{\text{rel}} = \frac{1}{M}\sum_{i=1}^{N} \mathbb{I}_{\{y_c^i = y^i \,\&\, y_{\text{cl}}^i = y^i\}}$$

where $\mathbb{I}_{\{y_{\text{cl}}=y\}} = 1$ if $y_{\text{cl}} = y$ and 0 else.

Methodology point: the clean accuracy $\mathscr{A}_{\text{cl}}$ is used to save the model's parameters during training; the absolute corruption accuracy $\mathscr{A}_c$ gives the model's accuracy for a corruption c; the relative corruption accuracy $\mathscr{A}_c^{\text{rel}}$ computes how many corrupted simulations x_c were correctly classified among the correctly classified clean counterparts; the positiveness of the rCE remains a relevant sanity check for debugging and verifying that the above hypothesis is preserved by the corruption simulator. By leveraging the above-mentioned structural principle for simulated corruptions, this metric addresses more accurately (in the statistical analysis sense) questions like "how do corruptions intrinsically behave for this model".

Our experiments show that $\mathscr{A}_c^{\text{rel}}$ doesn't overestimate the model's robustness and abnormal behavior[4] implies that a selected corruption simulation is ill-posed. On the other hand, the Relative mCE increasingly underestimates it, as highly accurate models will see a greater proportion of their miss-classified corruptions to come from miss-classified clean samples, making $\mathscr{A}_c$ to decrease while clean accuracy increases, as shown in Fig. 3 of Hendrycks and Dietterich [2019]. This phenomenon is confirmed in Figures 5–8 of Mu and Gilmer [2019] where miss-classified clean images represent 12% of the shown examples and none of them incur into well-classified associated corruptions.

C.3. Denoising Diffusion Probabilistic Models

In this section we present DDPMs in their analytic formulation, we derive their associated neural DEs and learning procedure giving a few examples of their numerical integration schemes. Then, we tackle the sampling speed problematic showing a trade-off between accuracy and speed among SDE & ODE solvers, the latter being faster but less accurate. Finally, we present the denoising method for purifying adversarial attacks and highlight the fact that ODE solvers, while faster than their SDE counterparts, are much more brittle to such attacks. This fact sheds evidence on the importance of constructing fast SDE solvers for diffusion-based purifiers.

C.3.1 Denoising Diffusion Probabilistic Models

Let $q_0(\mathbf{x}_0)$ be the unknown data distribution from which each sample $\mathbf{x}_0 \in \mathbb{R}^d$ is sampled. Injecting noise through a DDPM defines a forward diffusion process $\{\mathbf{x}_t\}_{t\in[0,T]}$ with $T > 0$ which

[4] Such as unexpectedly observing $\mathscr{A}_c > \mathscr{A}_c^{\text{rel}}$.

follows the stochastic differential equation (SDE):

$$\mathrm{d}\mathbf{x}_t \quad := \quad f(t)\mathbf{x}_t\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{\omega}_t, \quad t \in [0,T] \tag{C.4}$$

where $\boldsymbol{\omega}$ is a d-dimensional Brownian motion, and $f, g : \mathbb{R} \to \mathbb{R}$ are called the drift and diffusion coefficients of the diffusion process $\mathbf{x}_t$ respectively. These are designed so that the end-time process distribution $q_T(\mathbf{x}_T) \approx \mathscr{N}(\mathbf{x}_T|\mathbf{0}, \tilde{\sigma}^2\mathbf{I}_d)$ for some $\tilde{\sigma} > 0$, where $q_t(\mathbf{x}_t)$ denotes the marginal distribution of $\mathbf{x}_t$ at time t.

As an example, define the linear noise schedule $\beta(t) := \beta_{\min} + (\beta_{\max} - \beta_{\min})t$ between two chosen extrema $20 \simeq \beta_{\max} > \beta_{\min} \simeq 0$. Then the forward SDE, which is the first and non-neural component of a DDPM, reads

$$\mathrm{d}\mathbf{x}_t \quad := \quad -\frac{1}{2}\beta(t)\mathbf{x}_t\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\boldsymbol{\omega}_t, \quad t \in [0,T] \qquad q_0(\mathbf{x}_0) \sim p_{\text{data}},$$

As such, for any $t \in [0,T]$, the transition distribution of $\mathbf{x}_t$ conditioned on $\mathbf{x}_0$ satisfies

$$q_t(\mathbf{x}_t|\mathbf{x}_0) = \mathscr{N}(\mathbf{x}_t|\alpha(t)\mathbf{x}_0, \sigma^2(t)\mathbf{I}_d), \tag{C.5}$$

where $\alpha(t), \sigma(t) \in \mathbb{R}^+$ are differentiable functions of t with bounded derivatives, and we denote them as α_t, σ_t for simplicity. The choice for α_t and σ_t is referred to as the *noise schedule* of a DPM and is assumed that the *signal-to-noise-ratio* (SNR) α_t^2/σ_t^2 is strictly decreasing w.r.t. t. Variance-Preserving DPMs are such that $\sigma_t = \sqrt{1-\alpha_t^2}$ i.e. $\alpha_t^2 + \sigma_t^2 = 1$ for all $t \in [0,T]$ and $\tilde{\sigma} = 1$.

As a matter of fact, the reverse diffusion "denoising" process has been shown in Anderson [1982] to be driven by the explicit SDE

$$\mathrm{d}\mathbf{x}_t \quad = \quad [f(t)\mathbf{x}_t - g^2(t)\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)]\mathrm{d}t + g(t)\mathrm{d}\bar{\boldsymbol{\omega}}_t, \quad t \in [T,0] \tag{C.6}$$

where $\bar{\boldsymbol{\omega}}_t$ is a reverse-time Brownian motion and $\mathrm{d}t$ is a negative time-step. Among the different components of this reverse SDE, the gradient of the log-density (also named score function) $\log q_t(\mathbf{x}_t)$ is difficult to compute. Score-based generative learning is a framework in which one sets a neural network that learns to approximate this score function as we will see later on.

While the time t analytic solution to (C.4) w.r.t. an initial value $\mathbf{x}_0$ is given by

$$\mathbf{x}_t \quad = \quad \mathbf{x}_0 + \int_0^t f(s)\mathbf{x}_s\mathrm{d}s + \int_0^t g(s)\mathrm{d}\boldsymbol{\omega}_s,$$

one should be careful not to confuse forward and backward processes for the reverse SDE.

Remark C.3.1. The analytic solution to (C.6) formally given by

$$\mathbf{x}_t \quad = \quad \mathbf{x}_T - \int_t^T (f(s)\mathbf{x}_s - g^2(s)\nabla_{\mathbf{x}_s}\log q_s(\mathbf{x}_s))\mathrm{d}s - \int_t^T g(s)\mathrm{d}\bar{\boldsymbol{\omega}}_s, \tag{C.7}$$

where $t \to 0$ is a reverse time parameter, is equivalent to the following expression, for $\mathbf{y}_\tau := \mathbf{x}_{T-\tau}$,

$$\mathbf{y}_\tau \quad = \quad \mathbf{y}_0 - \int_0^\tau (f(T-s)\mathbf{y}_s - g^2(T-s)\nabla_{\mathbf{y}_s}\log q_{T-s}(\mathbf{y}_s))\mathrm{d}s - \int_0^\tau g(T-s)\mathrm{d}\bar{\boldsymbol{\omega}}_s, \tag{C.8}$$

where $\tau \to T$ is a forward time parameter.

In particular, the above Remark stresses the fact that the functions f, g in (C.7) are to be understood as going backwards-in-time.

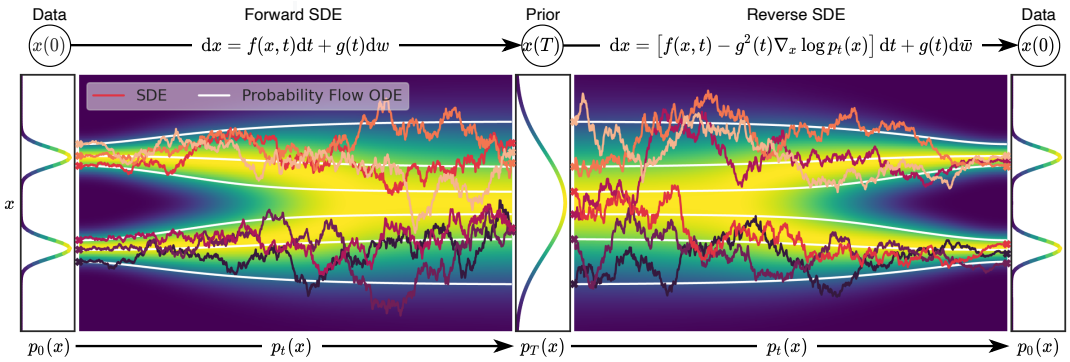


Figure C.2: Illustration of trajectories following the analytic system pre-determining a DDPM.

C.3.1.1 The Probability Flow ODE

A remarkable property of $q_t(\mathbf{x}_t)$ is that it follows also an associated ODE. Indeed, by the Fokker-Planck equation, the time-derivative of the marginal distribution q_t given by

$$\begin{aligned}\frac{\partial q_t(\mathbf{x}_t)}{\partial t} &= -\nabla_{\mathbf{x}_t}\cdot\left(f(t)\mathbf{x}_t q_t(\mathbf{x}_t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t} q_t(\mathbf{x}_t)\right)\\ &= -\nabla_{\mathbf{x}_t}\cdot\left(f(t)\mathbf{x}_t q_t(\mathbf{x}_t) - \frac{1}{2}g^2(t) q_t(\mathbf{x}_t)\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)\right)\\ &= \nabla_{\mathbf{x}_t}\cdot\left(\left(\frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t) - f(t)\mathbf{x}_t\right) q_t(\mathbf{x}_t)\right)\end{aligned}$$

which equals the following Liouville's equation

$$\mathrm{d}\mathbf{x}_t = \left[f(t)\mathbf{x}_t - \frac{g^2(t)}{2}\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)\right]\mathrm{d}t. \tag{C.9}$$

which will also be followed by $q_t(\mathbf{x}_t)$ and which we will call the probability flow ODE (PFO).

In Figure C.2 one illustrates the analytic mechanism encompassing what will be later defined as Denoising Diffusion Probabilistic Model (DDPMs): it takes a sample from the data distribution and gradually destroys its information until it becomes an easy-to-produce noise instance and then reverses this process to generate a sample from the original data distribution either by using the Reverse SDE or the PFO.

C.3.2 Score-based Modeling and Neural Differential Equations

Approximating $\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)$ by direct regression

$$\min_{\theta}\mathbb{E}_{t\sim\mathscr{U}(0,T)}\mathbb{E}_{\mathbf{x}_t\sim q_t(\mathbf{x}_t)}\|\boldsymbol{s}_\theta(\mathbf{x}_t,t) - \nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)\|_2^2$$

is not possible since the score $\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)$ of the marginal diffused density $q_t(\mathbf{x}_t)$ is not tractable. By writing

$$q_t(\mathbf{x}_t) = \int q_0(\mathbf{x}_0)q_t(\mathbf{x}_t|\mathbf{x}_0)d\mathbf{x}_0$$

we see that if we diffuse individual data points $\mathbf{x}_0$, the diffused $q_t(\mathbf{x}_t|\mathbf{x}_0)$ becomes tractable. Denoising score matching set

$$\min_{\theta} \mathbb{E}_{t\sim\mathscr{U}(0,T)} \mathbb{E}_{\mathbf{x}_0\sim q_0(\mathbf{x}_0)} \mathbb{E}_{\mathbf{x}_t\sim q_t(\mathbf{x}_t|\mathbf{x}_0)} \|s_\theta(\mathbf{x}_t,t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0)\|_2^2$$

which, after expectations, gives an approximation $s_\theta(\mathbf{x}_t,t) \approx \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$ which will be called *data prediction model*.

We can further simplify this problem by predicting only the noise to be added or retrieved to a previous step sample instead of predicting the whole noisy sample. More specifically, let $\mathbf{x}_t := \sqrt{\alpha_t}\cdot\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$, $\boldsymbol{\epsilon}\sim\mathscr{N}(\mathbf{0},\mathbf{I})$. Then

$$\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0) \quad = \quad -\nabla_{\mathbf{x}_t} \frac{(\mathbf{x}_t - \sqrt{\alpha_t}\cdot\mathbf{x}_0)^2}{2\sigma_t^2} = -\frac{\boldsymbol{\epsilon}}{\sigma_t}$$

so that we can define a *noise prediction model*

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t) \quad := \quad -\sigma_t s_\theta(\mathbf{x}_t,t)$$

that is trained by means of

$$\min_{\theta} \mathbb{E}_{t\sim\mathscr{U}(0,T)} \mathbb{E}_{\mathbf{x}_0\sim q_0(\mathbf{x}_0)} \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathscr{N}(\mathbf{0},\mathbf{I})} \frac{1}{\sigma_t^2} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\|_2^2.$$

In particular, the noise prediction framework approaches the scaled gradient of the log-density $-\sigma_t\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)$ by a neural network $\varepsilon_\theta(\mathbf{x}_t,t)$ so that the resulting model is formalized a neural stochastic differential equation

$$\mathrm{d}\mathbf{x}_t \quad = \quad \left[f(t)\mathbf{x}_t + \frac{g^2(t)}{\sigma_t}\varepsilon_\theta(\mathbf{x}_t,t)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\omega}_t, \tag{C.10}$$

which we will call reverse stochastic diffusion process (RSDP). The neural SDE formalism will then handle the discretization of the RSDP by a SDE solver of our choice. For instance, the Euler-Maruyama solver fixes a step-size h and defines the iteration rule from s to $t = s-h$ as

$$\mathbf{x}_t = \mathbf{x}_s - [f(s)\mathbf{x}_s + \frac{g^2(s)}{\sigma_s}\varepsilon_\theta(\mathbf{x}_s,s)]h + g(s)\sqrt{h}\varepsilon, \quad \varepsilon\sim\mathscr{N}(\mathbf{0},\mathbf{I}_d). \tag{C.11}$$

As such, DDPMs can be seen as neural differential equations (NDEs) for which we have at hand a strong inductive bias constraining them, seen as dynamical systems (DS), in the form of a probabilistic model. In other words, the diffusion SDE is a pre-specified continuous-time probabilistic model upon which we approximate one of its components by a neural network. As we argued in Gonzalez et al. [2022b], DPMs are instances of DS-destined neural differential equations.

Alternatively, one can model an equivalent problem via neural ODEs for the FPO:

$$\mathrm{d}\mathbf{x}_t := h_\theta(\mathbf{x}_t,t)\mathrm{d}t \quad = \quad \left[f(t)\mathbf{x}_t + \frac{g^2(t)}{2\sigma_t}\varepsilon_\theta(\mathbf{x}_t,t)\right]\mathrm{d}t, \tag{C.12}$$

for which we have a choice of fast but less accurate solvers. For instance the Euler solver iterates

$$\mathbf{x}_t = \mathbf{x}_s - h_\theta(\mathbf{x}_s, s)h.$$

As mentioned in C.3.1, the functions f, g are running backwards in time.

DPM-customized solvers: At a first glance, one can consider the PFO (C.9) as a neural ODE, in which case the problem of approximating it transforms into approximating the analytic solution

$$\begin{aligned}\mathbf{x}_t = \mathbf{x}_s - \int_s^t h_\theta(\mathbf{x}_\tau, \tau)\mathrm{d}\tau &= \mathbf{x}_s - \int_s^t \left(f(\tau)\mathbf{x}_\tau + \frac{g^2(\tau)}{2\sigma_\tau}\varepsilon_\theta(\mathbf{x}_\tau, \tau)\right)\mathrm{d}\tau \\ &\approx \texttt{odeint}(h_\theta, \mathbf{x}, s, t)\end{aligned}$$

As such, many works have proposed optimized solvers to speed-up this process by proposing either training-free solvers with relatively small improvements or solvers that need further training to reach acceptable speed Song et al. [2020], Nichol and Dhariwal [2021], Watson et al. [2021], Tachibana et al. [2021], Luhman and Luhman [2021], Kong and Ping [2021], Kingma et al. [2021], Jolicoeur-Martineau et al. [2021], Salimans and Ho [2022], Bao et al. [2022], Zhang et al. [2022]. Recently, in Zhang and Chen [2022], Lu et al. [2022] a further step was made in this direction. The idea at the heart of the construction of DPM-tailored solvers is to exploit the semi-linear structure of the PFO and simplify it to formulate a dedicated exponential ODE solver for it inspired by the construction of exponential RK methods. Their method starts by considering the general formulation of time-varying semi-linear problems Yang et al. [2022]

$$\mathbf{x}_t = e^{\int_s^t f(\tau)\mathrm{d}\tau}\mathbf{x}_s + \int_s^t \left(e^{\int_\tau^t f(r)\mathrm{d}r}\frac{g^2(\tau)}{2\sigma_\tau}\varepsilon_\theta(\mathbf{x}_\tau, \tau)\right)\mathrm{d}\tau.$$

Then, using the change of variables from Kingma et al. [2021], one can further accelerate this exponential method and rewrite the analytic solution $\mathbf{x}_t$ with initial value $\mathbf{x}_s$ as a sum of a linear term and an exponentially weighted integral of ε_θ as is done in Lu et al. [2022]. From this point, by developing the Taylor expansion of ε_θ and fitting t and s as two consecutive time-steps they develop three algorithms which they show to be, under natural assumptions to be solvers with orders one to three.

C.3.3 Purification pre-processing as a defense

A distinctive property of diffusion models is that they pull distinct data distributions near to each other. Formally, let $\{\mathbf{x}(t)\}_{t\in[0,1]}$ be a diffusion process defined by the forward SDE (C.4) and let p_t and q_t the respective distributions of $\mathbf{x}_t$ when $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$ and $\mathbf{x}_0 \sim q_0(\mathbf{x}_0)$. Then $p_t(x)$ and $q_t(x)$ are smooth and fast decaying, *i.e.*,

$$\lim_{\mathbf{x}_t^i\to\infty} p_t(\mathbf{x}_t)\frac{\partial}{\partial \mathbf{x}_t^i}\log p_t(\mathbf{x}_t) = 0 \quad \text{and} \quad \lim_{\mathbf{x}_t^i\to\infty} q_t(\mathbf{x}_t)\frac{\partial}{\partial \mathbf{x}_t^i}\log q_t(\mathbf{x}_t) = 0, \quad i = 1, \dots, d,$$

so that a straightforward integration by parts show that

$$\frac{\partial D_{KL}(p_t||q_t)}{\partial t} = -\frac{g^2(t)}{2}D_F(p_t||q_t) \leqslant 0$$

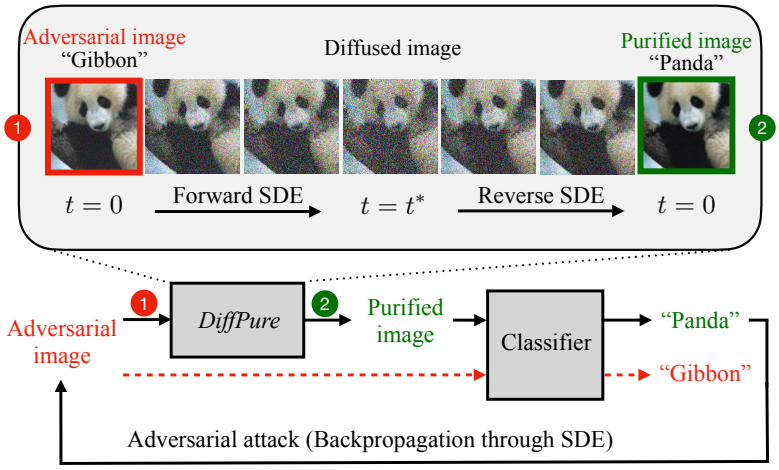


Figure C.3: The Purification framework in its adversarial defense form

where the equality happens only if the distributions p_t and q_t are the same. The repercussion of this phenomenon is that for each $\varepsilon > 0$, one can retrieve a minimum step t^* such that

$$D_{KL}(p_{t^*} || q_{t^*}) \leqslant \varepsilon.$$

The purification method will then consist on diffusing an attacked sample $\mathbf{x}_a$ up to t^* by following the SDE (C.4), denoising the resulting noisy sample $\mathbf{x}_{a,t^*}$ backwards to initial time, and classifying the resulting *purified* sample $\hat{\mathbf{x}}_a$ computed as

$$\hat{\mathbf{x}}_a = \texttt{sdeint}(\mathbf{x}_{a,t}, \bar{h}_\theta(\mathbf{x}_t, t), t^*, 0), \qquad \bar{h}_\theta(\mathbf{x}_t, t) := \left[f(t)\mathbf{x}_t + \frac{g^2(t)}{\sigma_t} \varepsilon_\theta(\mathbf{x}_t, t) \right] \mathrm{d}t + g(t) \mathrm{d}\bar{\omega}_t,$$

by a subsequent classifier $y_{\text{pred}} = f_{\text{clf}}(\hat{\mathbf{x}}_a)$. We will call t^* a purification time-stamp and one can show that retrieving it from the linear and cosine noise schedules can be done in a closed form.

The purification method using DDPMs has been used for defending against adversarial attacks both in empirical and certified approaches in Nie et al. [2022], Carlini et al. [2022]. On the one hand, empirical adversarial defenses find the purification timestamp from D_{KL} and run two neural SDEs: a first one acting as the purifier and a second one dedicated to back-propagating adaptive attacks through the adjoint SDE to efficiently obtain full gradients of the defense system. Additionally, by fine-tuning an adversarially trained classifier with the generated purified images incurs in further robustness improvements. On the other hand, certified adversarial defenses through denoised smoothing start from a distribution $\mathcal{N}(x, \sigma^2 \mathbf{I}_d)$ for a given noise level σ and directly compute $t^\star$ in a closed form as the solution to $\frac{1-\alpha_{t^\star}}{\alpha_{t^\star}} = \sigma^2$. Additionally, by exploiting the nature the noise encountered in adversarial attacks, they find that applying a one-shot denoiser to $\mathbf{x}_{a,t^*}$ shows better accuracy than iterative denoisers and is better suited for estimating robustness certificates that need sampling a large number of noisy instances.

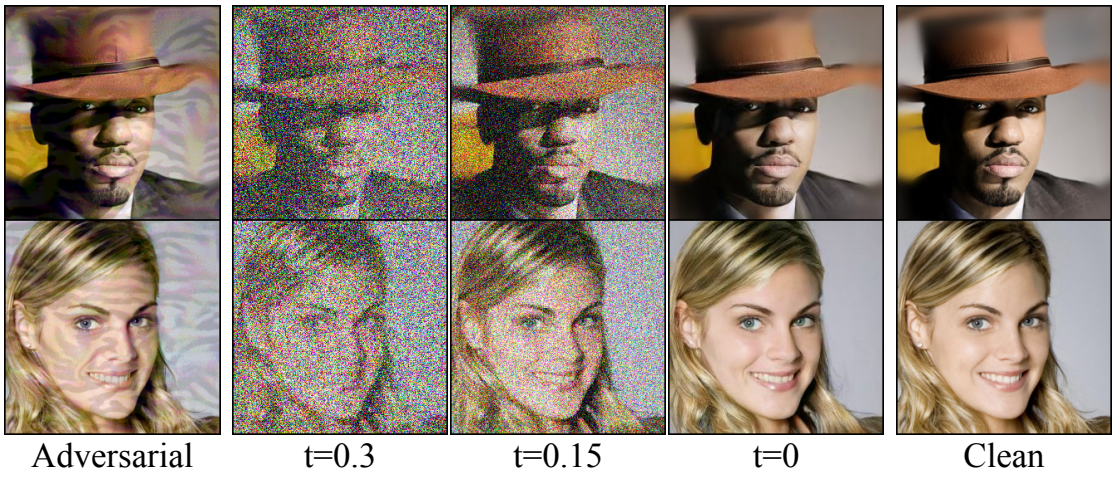


Figure C.4: Adversarial purification process on samples

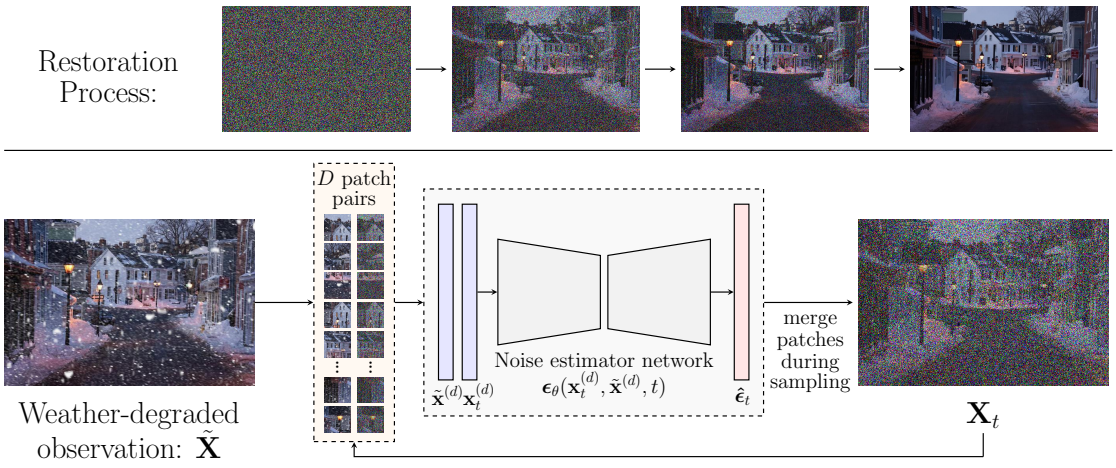


Figure C.5: Weather purification process from Özdenizci and Legenstein [2022]

C.4. Experiments

C.4.1 Noisy learning for Neural ODEs versus ResNets

Table C.1: Mean $\mathscr{A}_c^{\text{rel}}$ (%) on corrupted MNIST. A $> 5\%$ difference of model's performance is colored in orange. The last block computes the improvement on $\mathscr{A}_c^{\text{rel}}$ for each model induced by noisy training w.r.t. clean training. The listed corruptions are 1: gaussian, 2: shot, 3: impulse, 4: defocus, 5: glass, 6: motion, 7: zoom, 8: snow, 9: frost, 10: fog, 11: brightness, 12: contrast, 13: elastic_transform, 14: pixelate, 15: jpeg_compression.

Model	Training	$\mathcal{A}_{\text{cl}}$	$\mathcal{A}_c^{\text{rel}}$ on Noise			$\mathcal{A}_c^{\text{rel}}$ on Common Corruptions (severity 1)														
			Gaussian (σ)			Noise			Blur				Weather				Digital			
			50	75	100	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ResNet	Clean	99.45	98.1	88.9	66.3	99.7	99.8	98.8	79.9	35.9	96.9	99.4	98.1	97.9	58.9	99.7	97.5	67.2	98.8	99.8
ODENet		99.59	**99.2**	**89.2**	**74.1**	**99.9**	**99.9**	**99.4**	**82.7**	**57.3**	**98.1**	**99.8**	**99.5**	**99.3**	**93.4**	**99.9**	**99.5**	**77.9**	**98.8**	**99.9**
ResNet	Noisy	99.44	-	-	98.7	99.9	99.2	99.8	81.7	71.8	93.0	99.6	99.4	99.6	85.3	99.8	97.8	91.3	99.5	99.9
ODENet		99.59	-	-	**99.3**	**99.9**	**99.9**	**99.9**	**91.2**	**87.1**	**98.6**	**99.8**	**99.7**	**99.9**	**95.2**	**99.9**	**99.6**	**95.9**	**99.7**	**99.9**
ResNet	Noisy $\mathcal{A}_c^{\text{rel}}$- clean $\mathcal{A}_c^{\text{rel}}$				**32.4**	**0.2**	-0.6	**1**	1.8	**35.9**	-3.9	**0.2**	**0.7**	**1.7**	**26.4**	**0.1**	**0.3**	**24.1**	0.7	**0.1**
ODENet					25.2	0	**0**	0.5	**8.5**	29.8	**0.8**	0	0.2	0.6	1.8	0	0.1	18	**0.9**	0

Table C.2: Mean $\mathscr{A}_c^{\text{rel}}$ (%) at changes in corruption severity for MNIST. At fixed $(c,s) \in C$, each block (in green) contains results for cleanly trained ResNet (upper-left), ODENet (lower-left) and their noisy counterparts (upper-right, lower-right) The listed corruptions are as in Table 1. Performance shifts are colored in red. Corruptions where noisy training is not beneficial are colored in blue.

Sev.	1		2		3		4		5		6		7		8		9		10		11		12		13		14		15	
1	99.7	99.9	99.8	99.2	98.8	99.8	79.9	81.7	35.9	71.8	96.9	93.0	99.4	99.6	98.1	99.4	97.9	99.6	58.9	85.3	99.7	99.8	97.5	97.8	67.2	91.3	98.8	99.5	99.8	99.9
	99.9	**99.9**	**99.9**	**99.9**	**99.4**	**99.9**	**82.7**	**91.2**	**57.3**	**87.1**	**98.1**	**98.6**	**99.8**	**99.8**	**99.5**	**99.7**	**99.3**	**99.9**	**93.4**	**95.2**	**99.9**	**99.9**	**99.5**	**99.6**	**77.9**	**95.9**	**98.8**	**99.7**	**99.9**	**99.9**
2	99.4	99.9	99.6	99.9	96.3	99.6	**54.0**	51.4	33.1	68.8	83.0	75.5	99.1	99.6	91.5	98.3	84.0	98.8	45.8	74.4	99.4	99.7	91.7	93.3	37.4	70.7	98.6	99.5	99.7	99.9
	99.8	**99.9**	**99.8**	**99.9**	**98.1**	**99.7**	41.5	**58.9**	**50.3**	**84.5**	**88.3**	**92.1**	**99.7**	**99.7**	**98.0**	**99.5**	**94.6**	**99.7**	**86.6**	**85.5**	**99.8**	**99.9**	**99.0**	**99.3**	**50.7**	**82.9**	**99.1**	**99.6**	**99.9**	**99.9**
3	98.6	99.7	99.4	99.8	91.4	99.5	**16.1**	**18.4**	13.0	23.6	47.6	49.2	98.7	99.4	82.8	88.9	69.7	96.7	34.7	55.3	97.9	99.5	77.7	83.0	21.6	41.1	91.6	97.6	99.7	99.9
	99.5	**99.9**	**99.6**	**99.8**	**94.0**	**99.7**	7.6	11.1	**16.4**	**30.4**	**61.3**	**65.4**	**99.5**	**99.7**	**93.9**	**96.1**	**89.4**	**99.6**	**75.8**	**64.4**	**99.4**	**99.9**	**95.7**	**98.3**	**29.0**	**56.1**	**95.8**	**98.7**	**99.8**	**99.8**
4	93.7	99.6	98.1	99.4	68.1	98.7	**11.1**	**13.7**	12.6	20.7	24.7	29.7	98.0	99.2	71.3	79.2	68.5	96.8	30.6	50.2	93.0	99.2	**50.9**	59.7	16.9	26.9	64.2	84.6	99.6	99.8
	93.8	**99.7**	**98.6**	**99.6**	**77.1**	**99.4**	8.9	3.5	**15.2**	**25.4**	**35.0**	**41.5**	**99.3**	**99.5**	**98.5**	**91.1**	**89.4**	**99.5**	**69.7**	**58.8**	**97.4**	**99.8**	44.8	**95.0**	**21.2**	**37.0**	**79.2**	**91.6**	**99.7**	**99.9**
5	67.3	98.9	94.9	98.7	39.1	95.9	**9.9**	**11.2**	12.3	18.3	19.6	24.2	96.4	98.8	67.0	84.1	60.6	94.4	24.4	36.9	74.1	98.7	**25.9**	33.7	14.4	17.1	61.9	78.1	99.5	99.6
	76.2	**99.2**	**96.4**	**99.3**	**52.8**	**97.7**	9.6	4.9	**14.2**	**19.5**	**25.0**	**34.6**	**99.0**	**99.2**	**87.5**	**94.3**	**86.7**	**99.4**	**52.2**	**43.3**	**93.2**	**99.7**	15.2	**86.6**	**16.9**	**23.8**	**72.1**	**84.0**	**99.7**	**99.8**

We propose a minimalist, yet precise, comparative analysis on the robustness of a simple NODE (ODENet) and its discrete counterpart (ResNet) against simulated image corruptions. The chosen models are trained on MNIST with two methods: *clean training* is done on clean-only images; *noisy training* is conducted on a random combination of 50% of clean images and 50% of images added Gaussian noise with randomly chosen $\sigma \in \{50, 75, 100\}$. We train each model on three different random seeds, each trained model is then tested on 3 runs of corruption simulations and only report the mean of the resulting 9 tests in Tables 1 & 2. We report the mean of the 3 models clean accuracy $\mathscr{A}_{\text{cl}}$ used to save each model's parameters at which the rest of the tests are conducted.

Results: Table 1 shows that ODENet is consistently more robust than ResNet and that cleanly trained ODENet has less necessity of data augmentation to achieve good performances than ResNet do, as seen in the last lines of Table 1, make us conclude that they are naturally more robust than ResNet. This experimental result is compatible with those appearing in the test-time adaptive models literature Sun et al. [2020], Wang et al. [2021]. In light of the study in Gilmer et al. [2019] relating adversarial attacks as naturally appearing in the scope of common corruptions, this result can also be thought as a sanity check for determining that adversarial robustness for NODEs, contrary to what is hypothesized in Huang et al. [2022], might *not* come from obfuscated gradients. This fact seems to be further confirmed in Chu et al. [2022] although we have some reserves on their argument: increasing the time horizon of a NODE should, in our opinion, rather be linked to the model's resilience, roughly seen as the speed of convergence *after* an input perturbation, while robustness criteria usually focuses on the distances and positions of inputs incurring on invariance of a model's prediction. At increasing severity, as shown in Table 2, notice that, while ODENet is more robust than ResNet on most corruptions, their decay of robustness is bigger, shifting on some of the corruptions to ResNets as the best model. Nonetheless, the same shift occur at higher severity levels with noisy trained ODENets. Namely, for defocus_blur, on clean train mode, the shift was done at level 3 while at noisy train mode it was only done at level 4. Analogously, for contrast corruption, the shift at severity 4 on clean train mode was never reached on noisy train mode. This sheds evidence to the fact that noisy training for ODENets acts as a *robustness locus widening* i.e. that the robustness neighborhood of data points x become bigger with data augmentation. Notice that these robustness neighborhoods are *threat-model free*: they do not depend on the choice of a norm ball as is commonly considered on gradient-based defenses. Finally, noisy training made ResNet more vulnerable to corruptions 2 and 6 at severity 1 but this vulnerability got corrected at severity 3. This may suggest that partial information on the trade-off between accuracy and robustness may be captured by a notion of model's deterioration *resilience* whose rigorous study will be included in our upcoming extended study.

We give further details on the application of our methodology to the presented experiments: the chosen corrupt simulation algorithm is shown to be non-trivial along different tested datasets (no miss-classified clean images incur into well-classified corrupted counterparts); we do not include corruptions in our train or validation sets, networks share the same h_x and h_y modules; weight-tied ResNet blocks correspond to discretized NODE blocks. Since our model is not DS-destined, we do not conduct a numerical convergence test for the chosen Euler method.

C.4.1.1 Model specifications

All our models share the same FE and FCC modules and the RM modules consist on the same layers to which one either applies a residual connection (for ResNet) or the *odeint* function (for ODENet). In order to favor our ability to compare ResNets and ODENets, we fix the Euler method as our ODE numerical solver at time range $[0, 1]$ with 0.1 time steps which corresponds to ten weight-tied residual blocks. Finally, we use Group Normalization (GN) instead of Batch Normalization (BN) to ensure that the dynamics of the RM module truly correspond to an autonomous NODE.

We train all our models for 100 epochs, learning rate 0.001, milestones [30, 60, 90]; decay

Table C.3: The FE h_x and FCC h_y modules are identical for all our ResNet and ODENet models for MNIST, SVHN and CIFAR. The arguments of Conv2d are in order: the input channel, output channel, kernel size, stride and padding. Conv2dTime ensures time-dependence of the convolution component. The two arguments of the Linear layer represents the input dimension and the output dimension of this fully-connected layer.

Common Modules	Sequenced Layers
h_x	Conv2d(1, 64, 3, 1) + GN + ReLU Conv2d(64, 64, 4, 2) + GN + ReLU
h_y	AdaptiveAvgPool2d + Linear(64,10)
RM	Internal Layers (input x)
ResNet	out:=(Conv2d(64, 64, 3, 1,1) + GN + ReLU Conv2d(64, 64, 3, 1,1) + GN + ReLU) out +x
ODENet	f=(Conv2dTime(64+1, 64, 3, 1,1) + GN + ReLU +Conv2dTime(64+1, 64, 3, 1,1) + GN + ReLU) odeint(f, x, [0,1], Δt=0.1, Euler)

0.0005, L_2-penalty 0.2. Both models have around 142k parameters.

While, in (C.1), f formalizes a single layer's dynamics, it usually is taken in practice to be a composition of explicit functions that we pass to the ODE solver (a block). Using BN as a block component holds mini-batch information, which not only cannot be formalized as an autonomous ODE but may lead to gradient explosion at back-propagation. When comparing ODENet and ResNet blocks (without BN), one must ensure that each composite function for a NODE block is both stateful (has an implicit dependence on the integration time) and input-autonomous (does not present dependence or has leaked information of the rest of the samples) either at training, validation or testing. We use the ReLU function for practical purposes (increased performance) after checking that models trained with fully differentiable functions present the same behavior. We avoid taking into account in our analysis neither neural SDEs, which can genuinely be seen as continuous-time analog of noise injection robustifying methods, but whose inference is not deterministic and which do not take into account stateful BN layers, as they induce depth-varying architectures which do not have a clear discrete counterpart upon which one could establish a comparative analysis. Choosing the good basis function method for the latter to achieve competitive results Queiruga et al. [2021] was a long effort, according to the authors, and hasn't yet come close to state-of-the-art clean accuracy performances. In addition, BN which has been shown to be crucial to achieve state-of-the-art robustness performances, has also been shown to be a source of adversarial vulnerability and it is unclear if their stateful counterpart from Queiruga et al. [2021] will present or not this same behavior. Finally, our discretized NODE block is formulated in terms of weight-tied ResNet blocks and matches the function to which the residual skip connection is added with the one sent to the ODE solver and does not match to the total length of convolution blocks present in the architecture. For instance, appending 10 independent identical residual convolutional blocks does not correspond to passing a single convolutional block through a ODE solved with fixed 10 Euler steps as the state space of the NODE is controlled by only one set of convolutional block parameters.

C.4.1.2 Further Simulation Corrupted Dataset Experiments

We use several datasets and their synthetic corruptions by using the same simulation algorithm and selecting corruptions that make sense on all datasets. Noisy training is done with randomly added noise with $\sigma \in \{10, 15, 25\}$ for the SVHN dataset, and with $\sigma \in \{10, 15, 20\}$ for the CIFAR10 dataset. Our experiment's relative perturbed accuracy is given in Tables 4 and 5 for SVHN[5].

Table C.4: Mean $\mathscr{A}_c^{\text{rel}}$ (%) on corrupted SVHN images for ResNet and ODENet. The listed corruptions are as in Table 1. The last block computes the improvement on performance for each model induced by noisy training w.r.t. clean training. Corruptions where noisy training is not beneficial are colored in blue and a $> 5\%$ difference of model's performance is colored in orange.

Model	Training	$\mathcal{A}_{\text{cl}}$	$\mathcal{A}_c^{\text{rel}}$ on Noise			$\mathcal{A}_c^{\text{rel}}$ on Common Corruptions (severity 1)														
			Gaussian (σ)			Noise			Blur				Weather				Digital			
			10	15	25	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ResNet	Clean	94.3	47.2	29.3	20.1	84.7	83.2	81.9	93.7	93.8	95.3	98.9	77.4	88.3	50.7	99.2	97.5	66.6	97.7	97.2
ODENet		96.3	**63.7**	**44.0**	**31.7**	**92.3**	**91.4**	**91.1**	**95.3**	**96.3**	**96.8**	**99.3**	**83.4**	**91.8**	**59.3**	**99.6**	**98.8**	**84.7**	**98.9**	**98.1**
ResNet	Noisy	94.2	-	-	93.2	95.2	95.2	93.9	93.4	94.8	95.1	98.9	81.9	91.3	45.8	99.1	97.4	87.5	98.3	97.9
ODENet		96.2	-	-	96.2	**97.5**	**97.3**	**97.0**	**94.6**	**96.4**	**96.5**	**99.3**	**86.7**	**94.6**	**53.2**	**99.6**	**98.1**	**93.0**	**99.1**	**98.8**
ResNet	Noisy $\mathcal{A}_c^{\text{rel}}$- clean $\mathcal{A}_c^{\text{rel}}$				**73.1**	**10.5**	**12**	**12**	**-0.3**	**1**	**-0.2**	**0**	**4.5**	**3**	**-4.9**	**0.1**	**-0.1**	**20.9**	**0.6**	0.7
ODENet					64.5	5.2	5.9	5.9	-0.7	0.1	-0.3	0	3.3	2.8	-6.1	0	-0.7	8.3	0.2	0.7

Results: ODENet is consistently more robust than ResNet and the latter benefited more than ODENet from noisy training. This confirms our MNIST conclusions on the natural robustness of ODENet. Notice that noisy training makes the models slightly more vulnerable for some corruptions. At severity changes, noisy training acts here again as a robustness locus widening. Interestingly, for those corruptions where noisy training made the model more vulnerable at severity 1, noisy trained models see their vulnerability remain only on half of those corruptions at severity 5. Finally, as shown in orange, it seems that noisy training makes ODENet more resilient to corruption deterioration than it does to ResNet.

C.4.1.3 Further thoughts on corruption simulators and their metrics

Baseline Normalization: Usual corruption metrics such as the mCE^f and the relative mCE^f for a model f are computed with respect to a baseline (e.g. AlexNet) error in order to normalize it over those corruptions that are known to be particularly challenging. For $(c,s) \in C$, they are the average over all 15 corruptions labels c of the clean and corrupted top-1 error rates (averaged first across all 5 severity levels):

$$\text{CE}_c^f = \left(\sum_{s=1}^{5} E_{s,c}^f\right) \Big/ \left(\sum_{s=1}^{5} E_{s,c}^{\text{AlexNet}}\right), \quad \text{rCE}_c^f = \left(\sum_{s=1}^{5} E_{s,c}^f - E_{\text{clean}}^f\right) \Big/ \left(\sum_{s=1}^{5} E_{s,c}^{\text{AlexNet}} - E_{\text{clean}}^{\text{AlexNet}}\right) \tag{C.13}$$

[5]The results for CIFAR showing no novel or different behaviour than the one conducted for SVHN other than an overall drop in accuracy, we do not present them in this preliminary report.

Table C.5: Mean $\mathscr{A}_c^{\text{rel}}$ (%) at changes in severity. The listed corruptions are as in Table 1. Red color means a shift of best model's accuracy w.r.t. previous severity value. Corruptions where noisy training is not beneficial are colored in blue.

Model	Training	Sev.	Noise			Blur				Weather				Digital			
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ResNet	Clean	1	84.7	83.2	81.9	93.7	93.8	95.3	98.9	77.4	88.3	50.7	99.2	97.5	66.6	97.7	97.2
ODENet			**92.3**	**91.4**	**91.1**	**95.3**	**96.3**	**96.8**	**99.3**	**83.4**	**91.8**	**59.3**	**99.6**	**98.8**	**84.7**	**98.9**	**98.1**
ResNet		2	69.3	64.8	64.8	84.1	81.2	87.0	98.7	57.3	73.6	39.8	97.8	95.9	42.6	97.4	95.9
ODENet			**83.0**	**80.8**	**80.2**	**86.9**	**87.1**	**90.6**	**99.1**	**68.1**	**80.5**	**48.5**	**98.9**	**98.0**	**63.0**	**98.9**	**97.2**
ResNet		3	48.6	46.1	51.9	**46.2**	38.2	70.2	98.5	64.1	63.2	26.5	95.5	91.9	24.2	89.1	94.7
ODENet			**67.1**	**65.8**	**70.8**	45.6	**45.7**	**76.7**	**99.0**	**71.0**	**71.9**	**33.2**	**97.4**	**96.4**	**35.8**	**93.9**	**96.3**
ResNet		4	32.2	26.6	31.7	**28.0**	30.4	54.6	98.3	54.4	60.9	19.9	91.9	75.5	18.7	68.0	90.5
ODENet			**50.1**	**44.5**	**51.2**	21.2	**35.6**	**61.3**	**98.7**	**62.4**	**70.1**	**25.1**	**94.8**	**90.7**	**24.7**	**75.0**	**93.3**
ResNet		5	18.9	18.7	19.6	**21.3**	24.7	47.3	97.7	52.2	55.0	15.0	85.3	49.2	15.4	59.3	83.5
ODENet			**33.2**	**33.4**	**35.1**	13.6	**24.9**	**53.8**	**98.3**	**61.5**	**64.8**	**18.8**	**89.9**	**78.9**	**17.7**	**65.7**	**87.9**
ResNet	Noisy	1	95.2	95.2	93.9	93.4	94.8	95.1	98.9	81.9	91.3	45.8	99.1	97.4	87.5	98.3	97.9
ODENet			**97.5**	**97.3**	**97.0**	**94.6**	**96.4**	**96.5**	**99.3**	**86.7**	**94.6**	**53.2**	**99.6**	**98.1**	**93.0**	**99.1**	**98.8**
ResNet		2	90.1	89.1	88.0	83.7	85.3	86.4	98.7	64.5	79.9	36.0	98.0	96.0	74.2	98.3	97.0
ODENet			**94.8**	**94.0**	**93.7**	**85.4**	**90.0**	**89.8**	**99.1**	**73.8**	**86.1**	**42.4**	**99.0**	**96.7**	**84.8**	**99.0**	**98.1**
ResNet		3	80.3	79.6	82.0	44.4	44.2	70.2	98.6	66.5	71.6	24.4	95.9	92.3	51.4	94.3	96.2
ODENet			**88.7**	**88.6**	**90.2**	**45.6**	**54.4**	**76.1**	**98.9**	**72.6**	**80.2**	**29.6**	**97.5**	**93.6**	**66.7**	**96.7**	**97.4**
ResNet		4	66.0	61.7	66.4	**28.3**	34.4	54.0	98.3	56.6	70.2	19.3	92.4	76.4	37.8	82.6	93.2
ODENet			**78.9**	**75.5**	**79.9**	26.6	**43.2**	**60.0**	**98.7**	**63.0**	**78.9**	**25.0**	**94.8**	**83.5**	**51.0**	**88.0**	**95.1**
ResNet		5	47.6	48.9	50.2	**21.3**	25.4	47.3	97.7	56.9	65.0	15.3	86.3	47.3	26.5	74.8	87.7
ODENet			**63.1**	**65.0**	**66.4**	17.4	**30.0**	**52.0**	**98.3**	**65.1**	**73.9**	**19.3**	**89.9**	**66.2**	**34.0**	**80.7**	**90.9**

We do not normalize our metrics for two reasons. First, the mCE^f was proposed as an attempt to unify robustness bench-marking under a unique number across many models, which we do not do here. But most importantly, it has been shown that feature aggregating networks and deeper nets markedly enhance robustness. Thus, as NODEs consist on a paradigm shift of the notion of depth, which becomes adaptive even while testing, we find that testing such network against a baseline fixed depth network such as AlexNet could be harmful for our comparative analysis. This is somewhat concordant with the last recommendation in Croce et al. [2022] for generating adversarial attacks for adaptive test-time defenses. We do not average our metrics across severity levels to analyse their behaviour at increasing severity.

Corruption simulators: Instead of testing our models on static corrupted datasets (MNIST-C, CIFAR10-C, ImageNet-C), we run the exact same corruption simulator[6] on the datasets of all our experiments (and which is the same one used by the authors that proposed the above-mentioned static corrupted datasets). For simplicity, corruptions that were specially tailored for MNIST (such as zig-zag or canny edges) that only make sense to be conducted on that dataset will be left out from our comparative study, as well as fully formalizable corruptions (such as rotations, translations..) that one can use as auxiliary data augmentation techniques such as adversarial and S&P noise augmentations. This should be taken into account as a partial reproducibility issue: while the performance of the considered models should decrease when tested on compressed JPEG corrupted datasets such as ImageNet-C, the comparative results conducted in this work do not show any qualitative distinction (although the overall model's

[6] Available at https://github.com/bethgelab/imagecorruptions.

accuracies decrease). Also, our objective is not to propose an architecture capable of achieving state-of-the-art robust performances in either of the mentioned datasets. Our choice to fix a common corruption simulator for different datasets is somehow a model-driven choice. It has been shown that classification error patterns between robust models and those coming from human perception are fundamentally different. As such, focusing on understanding a model's behavior around simulated corruptions can be improved by fixing one simulator and generating corruptions among different datasets. This allows to release some part of the randomness of a generated simulation and prevents different data augmentation techniques to guess a corruption simulator's parameters, which in a sense can be seen as information leakage. By using the corruption simulator as a white-box component of our generated corruption dataset we hope this will promote better model's transferability to some degree.

C.4.2 Air Liquide Cylinder Counting: Desnowification by image purification

For the Air Liquide Cylinder Counting we create a baseline for the purification preprocessing method of *desnowification* i.e. purifying the presence of snow as a corruption in the image. The baseline used is from Ren et al. [2019] and consists, as in the experiment above, of a weight tied ResNet that gradually restores an image and can be seen informally as the Euler discretization of a particular neural differential equation that we will not explicit in this part. After preprocessing, we use YOLOv5 (large) for object detection. The objective here is to use this architecture as a baseline from which all ongoing works related to image restoration via diffusion models will be compared.

Figure C.6 illustrates the baseline purification framework Ren et al. [2019]. It consists on recursive procedure that unfolds a shallow ResNet and adds a recurrent layer to exploit the dependencies of deep features across stages.

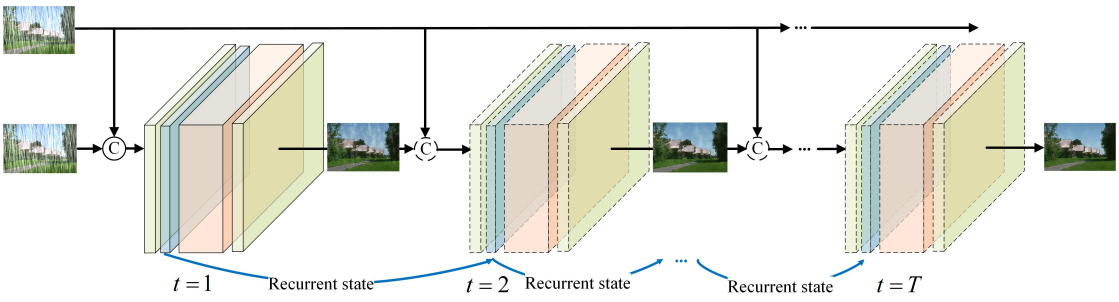


Figure C.6: Baseline framework Ren et al. [2019]

Image purifier training specifications:

We select 4 random images from each Air Liquide's folder 'WH BATCHES' (which contains around 40 videos), and we apply a fixed snow corruption of high intensity. The images are normalized between 0 and 1 then separated into 100x100 patches. Flipped versions of them are also considered during training to avoid overfitting.

On this dataset we train a PReNet (with a LSTM block) of width T=6, batch size of 18 and a learning rate of 1e-3 (decayed 3 times by a factor 5 through the training). The loss is the

Structural Similarity Index (SSIM) Wang et al. [2004] that assess perceptual quality differences between the original and purified image . The the optimizer is Adam and the model was trained for 100 epochs on two NVIDIA A100 GPUs of 32Gb each.

The trained PReNet purifies images which have been corrupted with the image corruption framework provided by Hendrycks and Dietterich [2019]. Three gradually increasing snow intensity levels (low, intermediate and high) are randomly applied following a uniform random distribution on the test set for validation purposes. Figure C.7 shows one example of the original image and its corruptions.

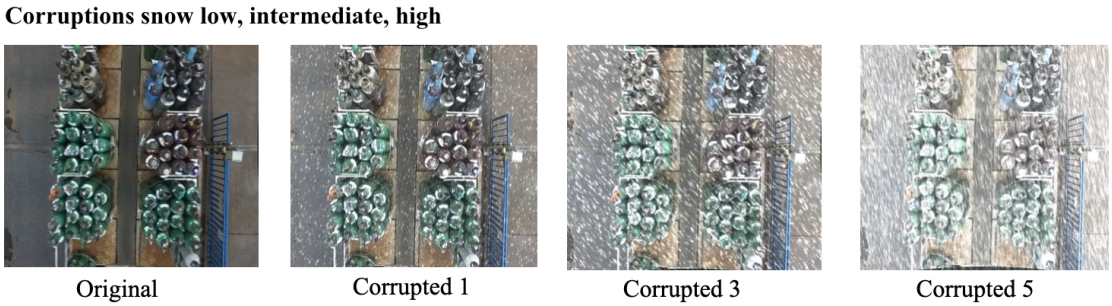


Figure C.7

Image purifier results:

Figure C.8 shows two examples of image purification from the test set in high and intermediate intensity. Perceptually speaking, the model seems to be subtracting the snow from the images as the recurrent forward steps (T) unfold. Small details of the bottles that were hidden by the corruption are now visible. This cleaning effect applies on different types of bottles regardless of its color and size. Also, we notice that the bottles are clearly distinguishable from the context (i.e floor, truck).

Figure C.9 shows that, as the purification unfolds, the average counting error shrinks from 3.1 bottles/truck to reach a minima of 1.18 and 1.05 for high and intermediate snow respectively. The counting error tends to stabilize at T=4 afterwards further iterations are not beneficial. The lower counting error of intermediate snow indicates that the purification task is easier. Comparing the obtained results with the reference clean data counting error of 0.951 bottles/truck, we conclude that the purifying model has not only provided a perceptual but an operational advantage compared to the classifier alone on corrupted data.

Low intensity performance: Now, we analyze how the purifier behaves in the presence of little to none amount of snow. Unlike the previous results, rendered images from Figure C.10 are overall darker and small details are not sharply visible. This is because, in the absence of white snowflakes the purifier attacks small white image features. In consequence, the counting error tends to increase as the purifying process unfolds until it stabilizes. Hence, exhibiting a worse performance than the classifier alone as evidenced in figure C.11 and table C.6.

Table C.6 shows a resume of the mean bottle counting error per truck for each snow level and purifying intensity. In the case of little to none snow, it makes sense not to apply the purification because the classifier alone accuracy is higher. The major advantage of the purifier is evidenced in intermediate and high snow events, showing a massive error reduction up to 10.4 bottles/truck.

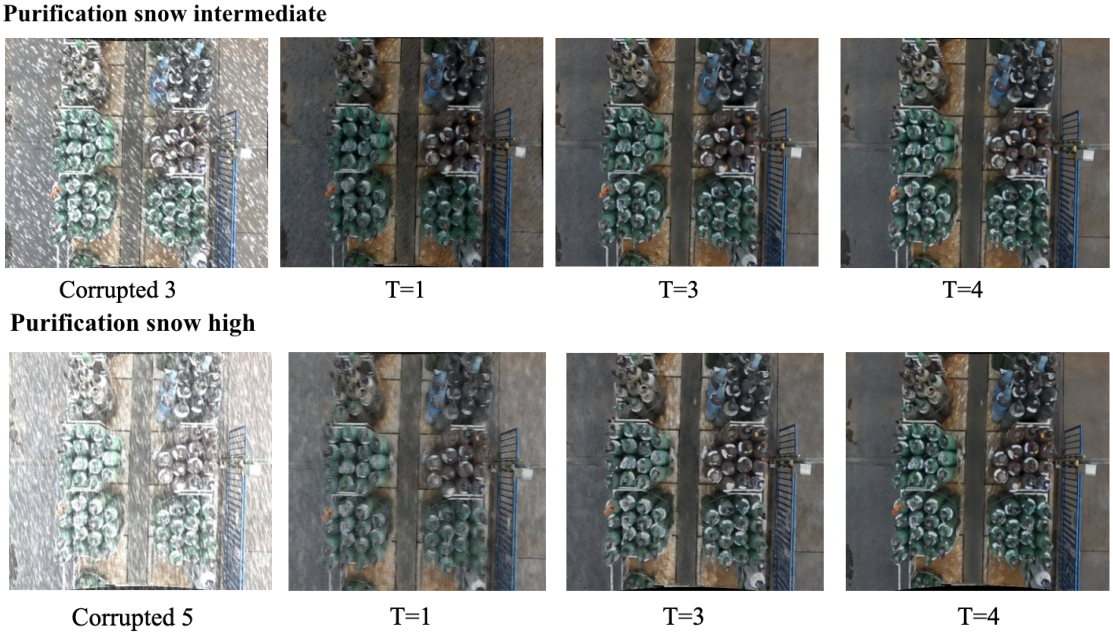


Figure C.8

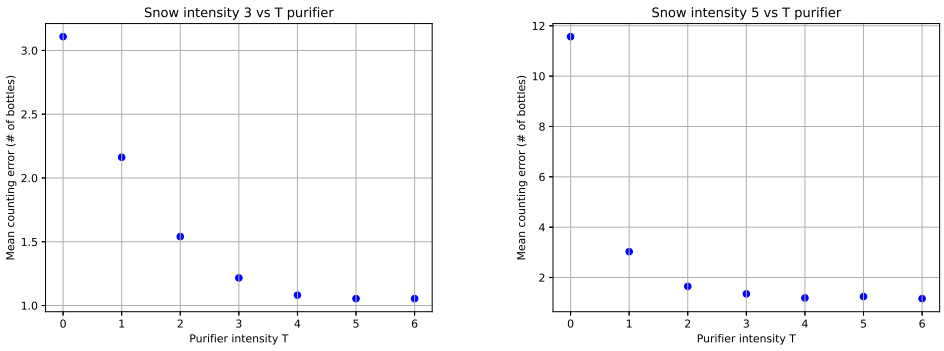


Figure C.9

Snow level\stage	0	1	2	3	4	5	6
Clean	**0.95**	7.48	10.13	8.16	7.37	6.83	6.29
Low	**1.24**	1.94	2.0	1.51	1.40	1.29	1.32
Intermediate	3.10	2.16	1.54	1.21	1.08	**1.05**	1.05
High	11.56	3.02	1.64	1.35	1.18	1.24	**1.16**

Table C.6: Mean counting error of the pipeline Counter+Purifier at increasing stage width of the purifier and for increasing levels of snow severity.

Conclusions: As we can see, the purification procedure shows a very different behavior in

low intensity snow events, imposing for the user the need to monitor the snow intensity before applying the purification procedure. This weather monitoring can be achieve either using supervised or unsupervised learning. A simple yet effective way of detecting this is to check whether the mean counting error increases in the first step or not. Further development on this subject will be discussed on the following section. Additionally, one can see that the purified images are somewhat darker that the original ones, specially at low width levels. As brightness and contrast of an image has been shown to be standard image corruptions, the fact of obtaining bad counting performances seems to be rather a natural phenomenon. Nonetheless, the reason why these images seem darker while their corresponding counterparts at intermediate and high corruption intensity do not become darker remains to be subject for further analysis.

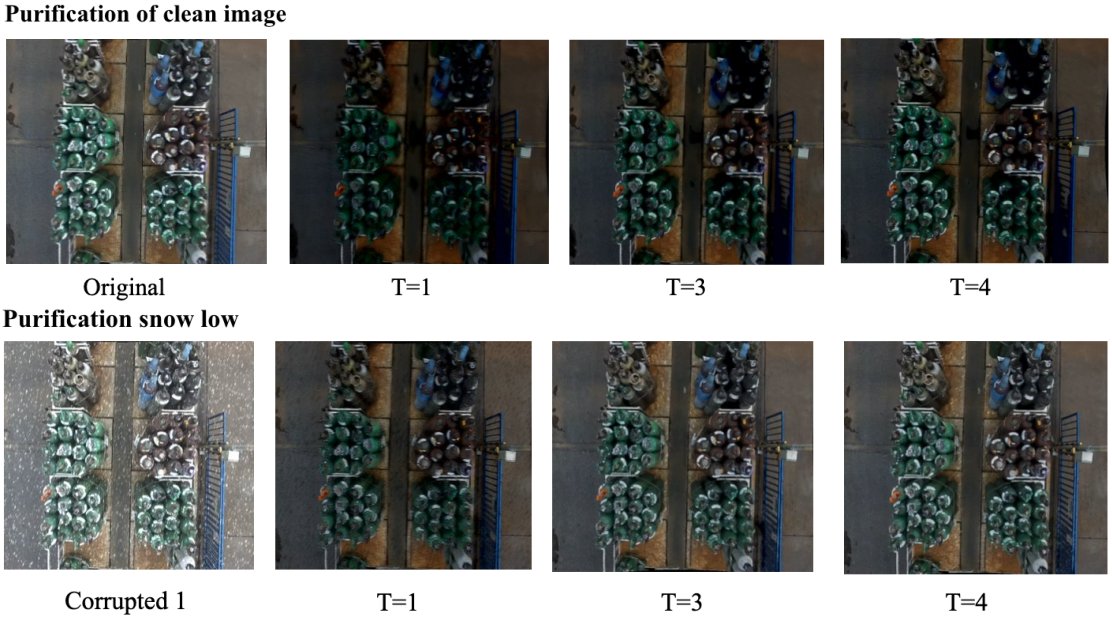


Figure C.10

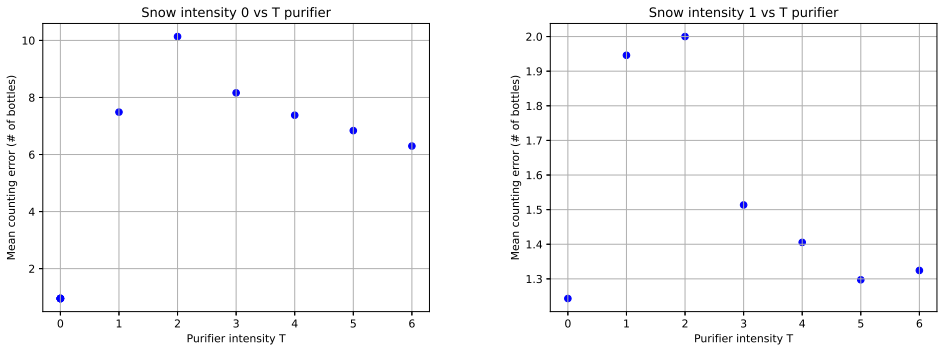


Figure C.11

C.4.2.1 Real-time weather corruption monitoring

In light of the above, we developed a lightweight, real-time snow monitoring that is able to detect snow events and their intensities. After careful examination, we use a supervised learning solution, profiting the data corruption pipeline previously created for the image purifier.

Training specifications:

We trained an image classifier type ConvMixer, firstly proposed by Trockman and Kolter [2022]. This architecture is compelling due to its high performance and low inference time. As for the input, we reduce the image size from 992x1024 by a factor of 1/4. In a second, step, we crop a centered region of 90x90 pixels. The training settings are the following: 5 epochs, a batch size of 8 images, Adam optimizer with a learning rate of 0.01 and weight decay of 0.01. The model's output are four classes: Clean, snow low, intermediate, and high in equal proportions. To avoid over-fitting, we dynamically change the corruption level of images during training and validation. Doing so, we make sure the model learns the snow distribution and does not focus on irrelevant contextual features. Figure C.12 illustrates the overall structure of the monitor. After training, the model reaches a 81.7% accuracy in a test set of 42 videos.

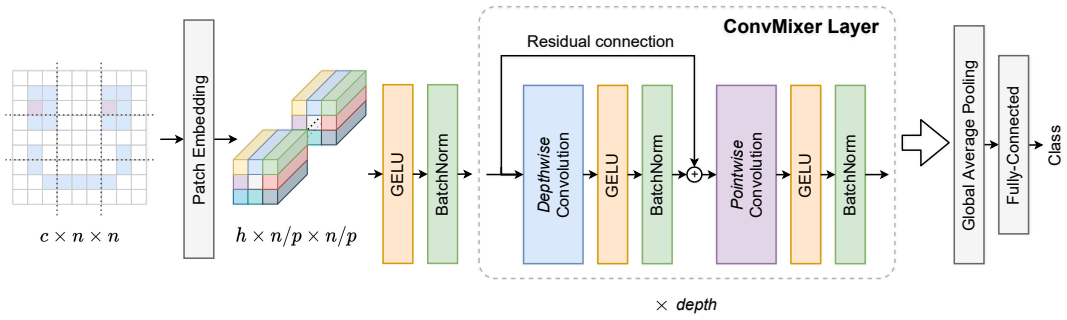


Figure C.12: ConvMixer from Trockman and Kolter [2022]

Figure C.13 shows an example of real-time snow intensity monitoring using the trained ConvMixer. Predictions are consistent for intermediate and high intensity with probabilities of 82% and 66% respectively. Nevertheless, the model has some difficulties to correctly identify the clean image, labeling it as snow low. On this class, the classification task becomes harder because the corruption does not alter the original image in a significant way.

C.4.2.2 Weather-robust cylinder counting pipeline

We propose a weather robust cylinder counting pipeline that includes a real-time snow detector and a snow purifier that dynamically cleanses the image only when needed. Based on the previous results, the purification initiates only when the image is classified as intermediate and high level of snow. For little to none snow, the monitoring model sends the image directly to the

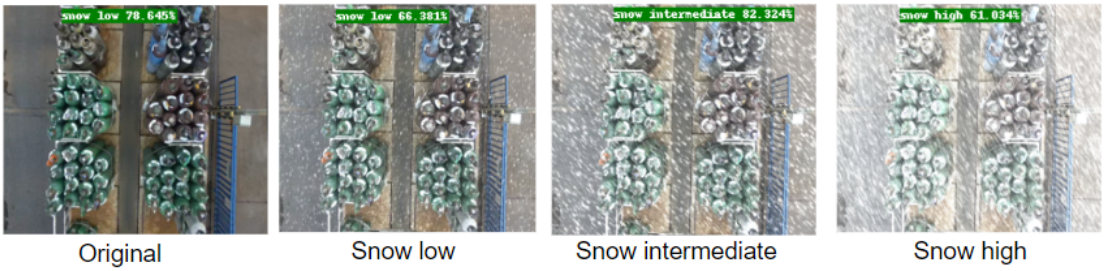


Figure C.13: Examples of snow detection of several intensities.

YOLOv5 counting classifier. By doing do, we save inference time and optimize the pipeline's overall performance. Figure C.14 shows the proposed snow purification pipeline.

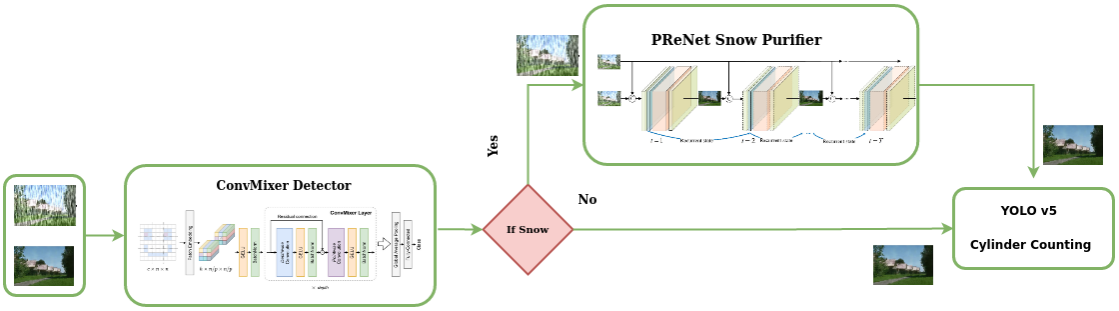


Figure C.14: Proposed weather-robust cylinder counting pipeline.

We then conduct the following tests. First, we provide the reference YOLOv5 mean counting error and its inference time. Next, we consider a corrupted data set consisting of 10 clean videos, 3 collections of 10 videos with snow at intensity 1,2 and 5 respectively. On this corrupted set we evaluate the performances of YOLOv5 alone, of YOLOv5 with a PReNet(t=4) Purification preprocessing at t=4 stages and lastly of YOLOv5+PReNet(t=4) first passed through the trained ConvMix which chooses if the amount of snow is enough for the image to necessitates a purification. In particular, the ConvMix sends the image to PReNet(t=4) if the snow intensity is greater or equal to 3.

Results: Table C.7 summarizes the obtained results.

Model	Dataset	Mean Counting Error	Inference time(s)
(ref) YOLO	clean	0.9512	0.06
YOLO	corrupt	3.4358	0.06
PReNet+YOLO	corrupt	1.9743	0.13
ConvMix+PReNet+YOLO	corrupt	**1.1784**	0.131 (mean)

Table C.7: Robust Benchmark on Air Liquide Cylinder Counting

We obtain good accuracy for both the high and low snow intensity regimes with a very low impact on the inference time. Nonetheless, it seems impossible to go below the mean counting

error threshold of 1 with the proposed baseline purifier as it never actually reaches that level across different corrupted datasets and number of stages.

C.5. Conclusion and Perspectives

We demonstrated the superiority of robustification approaches for neural networks based on neural differential equations or their discretized versions by a multitude of approaches. First of all, their analytic formulation allows to use a big and rich literature on stability of differential equations in order to infer ML robustness properties. In particular, by means of an *intrinsic robustness metric* whose benefits were shown in the above section, we established in a rigorous way ResNet-based baselines from which to create solid comparative analyses whose conclusions show the superiority of Neural DE and diffusion approaches for robustification in terms of quality. Under the Purification approach, we established an useful baseline to retrieve and purify snow corruption using the Air Liquide's Cylinder Counting use case. Our approach manages an average of 1.13 miscounted bottles across all videos with a minimal increment of inference time. We believe that our baseline method can readily be generalized to other meteorological corruptions with different levels of intensity, including rain, night and fog. Nevertheless, the proposed pipeline does not reduce counting error below the ideal threshold of 1 miscounted bottle per truck. In order to reach that performance, better image restoration approaches such as diffusion-based weather purifiers are needed Özdenizci and Legenstein [2022].

Future work: Diffusion-based purifiers vastly outperform the restoration capacities of PReNets and provide protection against unknown corruptions. Nevertheless, the current state of the art exhibits a prohibitively slow inference time. We currently explore new techniques that allow an important inference time speedup by means of the following steps:

- Passing from the PyTorch framework to the JAX framework is shown to cut all model inferences by half.
- Vectorizing the training procedure has been shown to reduce the training time by a factor of 10.
- Performing the diffusion on the latent space instead of the pixel space importantly reduce inference time Rombach et al. [2022].
- The use of exponential integrators that fasten inference time by a factor of 10 without compromising image quality Lu et al. [2022].

Additionally, further exploration on the generalization capabilities of the baseline purifier and its diffusion-based counterparts under real adverse weather conditions is relevant to ensure a future operational deployment.

Chapter D

Conformal prediction for time series

Foreword

[MAKE AS A FOOTNOTE]. Some sections of this chapter are either verbatim copies or adapted from existing texts previously published by the authors. This will be made explicit in the footnotes.

Contents

D.1. Introduction

This chapter presents an application of *Conformal Prediction* (CP) to time series data, specifically to the use case of "Air Liquide Demand Forecasting". CP is a **distribution-free** approach to *Uncertainty Quantification* (UQ) in *Machine Learning* (ML), built around one main hypothesis: the data samples are exchangeable [Aldous, 1985] or *independent and identically distributed* (i.i.d.), which is a special case of exchangeability. This allows great flexibility, as any model can be "*conformalized*", even if its performance is poor or it is misspecified; the CP procedure will yield valid prediction intervals, for any sample size (see Section D.3 for more details). However, the time series analysis deals with time-indexed sequential data, which are inherently not exchangeable. As we show in Section D.3.1.3, researchers had to find a trade-off between the convenience of distribution-free model-agnostic CP and the guarantees attainable, such as assuming i.i.d. or "well-behaved" prediction errors, to get some of the conformal guarantees.

D.2. Uncertainty quantification with conformal prediction

The work on CP described in this chapter is the continuation of what was carried out during 2021 and the first quarter of 2022, for the action sheet No. 5 of project EC3, within the program *Confiance.ai*. In the ensuing report [Nabhan et al., 2022], we introduced the principles of *Conformal Prediction* (CP), some of the most important algorithms, and an application to the Demand Forecasting[1] use case built by Air Liquide.

To make this exposition self-contained, we compiled a compendium on CP with texts from our paper [Mendil et al., 2022], which covers part of the work carried out in the project and which was published in the proceedings of *COPA 2022*[2,3].

Sections D.2.1, D.2.2 and D.2.3 are adapted from Section 2 in Mendil et al. [2022]. The *incipit* of Section D.3 and Section D.3.1 are adapted from Section 3 in Mendil et al. [2022]. The opening of Section D.5.1, Section D.5.1.1 and D.5.1.2 are adapted from Section 1 in Mendil et al. [2022].

D.2.1 Prediction intervals

Let $(X,Y) \sim \mathbb{P}_{XY}$, with X being the (random) vector of features and $Y \in \mathbb{R}$ the target in a regression learning task. Let $\alpha \in (0,1)$ be a **miscoverage** level[4], interpretable as the proportion of

[1] https://wiki.confiance.ai/wiki/Air_Liquide_Demand_Forecasting_UC_Guidelines

[2] https://proceedings.mlr.press/v179/mendil22a.html

[3] Mendil, Mossina, Nabhan, and Pasini [2022] is published under license *Attribution 4.0 International (CC BY 4.0)*, which allows to "[...] remix, transform, and build upon the material for any purpose, even commercially. [...]" (https://creativecommons.org/licenses/by/4.0/). For more information, see the publisher's license agreement: http://proceedings.mlr.press/pmlr-license-agreement.pdf.

[4] The reader can find some papers that accept the edge cases of $\alpha = 0$ and $\alpha = 1$, but in our experience this is for computational convenience. Statistically speaking, since we are working with random variables, setting $\alpha = 0$ would correspond to saying "we want zero errors with certainty", which doesn't make sense. Anytime the user will be faced with random event (always), they will have to find a trade-off between the acceptable error rate and the precision (i.e. size) of their prediction intervals.

predictive mistakes we are willing to accept in the long run, that is, when the predictive system is in operation. A *Prediction Interval* (PI) [Hahn and Meeker, 1991] obtained via algorithm $C_\alpha(X)$ should contain the true value of the random variable Y, $(1-\alpha)100\%$ of the time. Ideally, $\mathbb{P}\{Y \in C_\alpha(X) \,|\, X = x\} = (1-\alpha)$ holds true conditionally on the value taken by X. An inferential procedure that satisfies this statement is said to produce conditionally **valid** intervals[5]. When the weaker condition $\mathbb{P}\{Y \in C_\alpha(X)\} = (1-\alpha)$ holds, the procedure is **marginally** valid, on average over X.

D.2.2 Construction of prediction intervals

Let $Y \in \mathbb{R}$ be a prediction target, such as the demand of product at any given time in our industrial use case. There are two main approaches to build the interval $\widehat{C}_\alpha(X) = [\widehat{Y}^L, \widehat{Y}^U]$, with lower bound $\widehat{Y}^L$ and upper bound $\widehat{Y}^U$. First, one can add an error margin δ_α to a point prediction: $\widehat{C}_\alpha(X) = [\widehat{f}(X) - \delta_\alpha(X)\,,\, \widehat{f}(X) + \delta_\alpha(X)]$. Here, $\widehat{f}(\cdot)$ usually estimates the conditional expected value $\mathbb{E}\{Y \,|\, X = x\}$, and $\delta_\alpha(X) \geq 0$ depends on the data and a *probabilistic statement* made within the PI procedure; in linear regression, for instance, we could assume the errors to be normally distributed [Wasserman, 2004]. Second, one can estimate the bounds directly, e.g. by quantile regression: $\widehat{C}_\alpha(X) = [\widehat{q}_{\alpha_{lo}}(X)\,,\, \widehat{q}_{1-\alpha_{hi}}(X)]$, where $\widehat{q}_\beta(\cdot)$ is an estimation [Koenker and Bassett Jr, 1978, Meinshausen, 2006] of the conditional quantile:

$$q_\beta(x) = \inf\{y : F(y \,|\, X = x) \geq \beta\} \tag{D.1}$$

with $\alpha_{lo} + \alpha_{hi} = \alpha$. If valid, these intervals are *representative of the predictive error* due to the predictor (epistemic uncertainty) and the randomness in sampling (aleatoric uncertainty). Independently of the choice of $f(\cdot)$ or $q(\cdot)$, in Section D.3 we show that conformal inference builds valid intervals for *any method* and can "robustify" existing PIs with rigorous probability guarantees.

D.2.3 Metrics for prediction intervals

In the literature [Pang et al., 2018, Bazionis and Georgilakis, 2021], we find **coverage metrics** to assess how often the predicted intervals $\widehat{C}_\alpha(X)$ *contain* the true value of Y, and **sharpness metrics** to measure the *width* of the intervals. Coverage and sharpness are connected, as a degradation in one usually leads to an improvement of the other. Let n be the number of samples in a test dataset $D_{test} = \{(X_i, Y_i)\}_{i=1}^n$, $(X_i, Y_i) \sim \mathbb{P}_{XY}$. For a miscoverage level $\alpha \in (0,1)$, we get the corresponding **nominal** (target) **coverage** as $1-\alpha$. We measure how many times the PI contains the true value via the empirical coverage, or *Prediction Interval Coverage Probability*:

$$\text{PICP} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{\widehat{Y}_i^L \leq Y_i \leq \widehat{Y}_i^U\} \in [0,1]. \tag{D.2}$$

[5]This guarantee is on the *procedure* followed to sample the data and construct the PIs. The interval has a probabilistic interpretation only *before* observing the data. For a detailed explanation, see [Shafer and Vovk, 2008, Section 2.2] and [Hahn and Meeker, 1991, Section 2.3.6].

When PICP $\approx 1-\alpha$, the PIs are capturing, or "covering", the values of Y at the specified rate α. For a PICP $> 1-\alpha$ we have **overcoverage** and we are capturing too many points, for PICP $< 1-\alpha$, we have **undercoverage**.

In CP, the reference metric for sharpness is the *Average Width*:

$$\text{AW} = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i^U - \widehat{Y}_i^L). \tag{D.3}$$

The *Prediction Interval Normalized Average Width* allows for comparisons over different datasets, via the normalization $R = Y_{max} - Y_{min}$:

$$\text{PINAW} = \frac{1}{nR}\sum_{i=1}^{n}(\widehat{Y}_i^U - \widehat{Y}_i^L). \tag{D.4}$$

D.3. Algorithms for Conformal Prediction[6]

Many popular ML algorithms do not offer Uncertainty Quantification (UQ) out of the box [Hastie et al., 2009, Goodfellow et al., 2016], and some UQ methods are valid only under hypotheses on $\mathbb{P}_{XY}$ or asymptotic properties of the algorithm (e.g. Section D.2.1): they require enough data to get reliable PIs and can suffer from overfitting.

CP [Vovk et al., 2005] is a set of *distribution-free*, *model-agnostic* and *non-asymptotic* methods to do UQ by constructing valid *prediction sets* or *intervals*, whose probability coverage is backed by theoretical guarantees. Given a miscoverage probability $\alpha \in (0,1)$, a set of exchangeable[7] calibration[8] data $\{(X_i, Y_i)\}_{i=1}^n$ and test point (X_{new}, Y_{new}) with common distribution $\mathbb{P}_{XY}$, a CP procedure $C_\alpha(\cdot)$ builds prediction sets so that:

$$\mathbb{P}\Big\{Y_{new} \in C_\alpha\left(X_{new}\right)\Big\} \geq 1-\alpha. \tag{D.5}$$

Over many calibration and test sets, $C_\alpha(X)$ will contain the observed values of Y with frequency of *at least* $100(1-\alpha)\%$. Within the CP framework, Equation D.5 holds for any model, any data distribution $\mathbb{P}_{XY}$ and any finite sample sizes. Unlike the probability statement in Section D.2, this holds for Y when averaging over all possible X, that is, **marginally** and not conditionally [Foygel Barber et al., 2020]. We could overcover or undercover Y for some values of X.

CP can act as a *complementary tool* to attain rigorous probability coverages, as it can "conformalize" any existing predictor during or after training (black box predictors), yielding marginally valid PIs even under model misspecification [Chernozhukov et al., 2021]. Furthermore, any theoretical property of the underlying predictor still holds[9].

[6] The *incipit* of Section D.3 and Section D.3.1 are adapted from Section 3 in Mendil et al. [2022]

[7] This includes independent and identically distributed data (iid) as a special case [Aldous, 1985].

[8] In the original version of CP, also known as *Full Conformal Prediction* [Saunders et al., 1999, Vovk et al., 2005], the prediction problem is framed as online *transductive inference*, where the exchangeability is between the past observations and the test point X_{n+1}. This version is not computationally viable, throughout the text we will introduce versions on CP that rely on holdout "calibration" data, to "conformalize" our predictor.

[9] e.g., the asymptotic conditional coverage of Quantile Regression Forests [Meinshausen, 2006].

D.3.1 Related Work

In this section, we present the most common CP methods of the literature. The reader is also referred to Angelopoulos and Bates [2021] for a hands-on introduction to CP and Shafer and Vovk [2008], Zeni et al. [2020] for an in-depth overview. Let $D_{train} = \{(X_i, Y_i)\}_{i=1}^{n_{train}} \sim \mathbb{P}_{XY}$ be the training data and $\alpha \in (0,1)$ the miscoverage probability.

D.3.1.1 The split conformal prediction procedure

Following the *Split* CP method [Papadopoulos et al., 2002, Lei et al., 2018], also known as *Inductive* CP, here are the fundamental steps of conformalization:

(Step 1) Choose (or receive) a base prediction model: $f(\cdot)$

(Step 2) Choose a nonconformity score: $R = s(\widehat{f}(X), Y)$

(Step 3) Choose a data scheme: split training data D_{train} into two partitions, a *fit* subset D_{fit} and a *calibration* subset $D_{calibration}$

(Step 4) Fit: compute $\widehat{f}(\cdot)$ on D_{fit}

(Step 5) Calibrate: compute scores $\bar{R} = \{R_i\}$, $i = 1, \ldots, |D_{calibration}|$ on $D_{calibration}$

(Step 6) Get error margin $\delta_\alpha^f = (1-\alpha)(1 + \frac{1}{|D_{calibration}|})$-th empirical quantile of $\bar{R}$

(Step 7) Inference: build CP interval $\widehat{C}(X_{new})$, for observation X_{new}

The **base prediction model** $f(\cdot)$ (Step 1) is either a pre-fitted black-box $\widehat{f}$, or is selected through empirical experimentation or for operational needs. The selection of **nonconformity score** (Step 2) is at the heart of CP: here, the default score is the absolute deviation $R_i = |Y_i - \widehat{f}(X_i)|$ (Section D.3.1.2 for other possible scores). The split **data scheme** (Step 3) used to fit and conformalize a predictor works well with large datasets. We partition D_{train} in two disjoint sets: D_{fit}, to fit the predictor (Step 4), and $D_{calibration}$ for calibration. For smaller datasets, better options exist (Section D.3.1.2). For **calibration** (Step 5), the data scheme determines the out-of-sample evaluation of $\widehat{f}$: here, we compute the scores $\bar{R} = \{R_i\}_{i=1}^{|D_{calibration}|}$ on $D_{calibration}$. In (Step 6), we compute the error margin δ_α^f as a quantile of the sorted scores; δ_α^f is a constant and does not depend on the test point X_{new}. At **inference** (Step 7), we get the PI as:

$$\widehat{C}_\alpha(X_{new}) = \left[\widehat{f}(X_{new}) - \delta_\alpha^f \, , \, \widehat{f}(X_{new}) + \delta_\alpha^f\right]. \tag{D.6}$$

When conformalizing a pre-trained predictor $\widehat{f}$, one only needs additional data $D_{calibration} \sim \mathbb{P}_{XY}$, skipping Step 3 and the fit in Step 4.

D.3.1.2 Conformal regression methods

In the CP literature, different methods stem from the choice of nonconformity score and data scheme. Split CP turns into *Locally Adaptive Conformal Prediction* (LACP) when using scaled

scores $R_i = \frac{|Y_i - \widehat{f}(X_i)|}{\widehat{\sigma}(X_i)}$, where $\widehat{\sigma}(X)$ is a predictor of dispersion learned on D_{train}. This was proposed by Papadopoulos et al. [2002] and further studied by Papadopoulos et al. [2011], Papadopoulos and Haralambous [2011], Johansson et al. [2014], Boström et al. [2017], for different types of predictors. Here, we follow Lei et al. [2018] and $\widehat{\sigma}(X_i)$ predicts the conditional *Mean Absolute Deviation* (MAD) of $\left(Y - \widehat{f}(X)\right)$, conditioned on $X = x$. LACP is "local" in the sense that, for a point $(X_i, Y_i) \sim P_{XY}$, the prediction interval size will be corrected to represent the variability at $Y_i|X_i = x_i$. The PI is given by:

$$\widehat{C}_\alpha(X_{new}) = \left[\widehat{f}(X_{new}) - \widehat{\sigma}(X_{new})\,\delta_\alpha^{f,\sigma}\,,\, \widehat{f}(X_{new}) + \widehat{\sigma}(X_{new})\,\delta_\alpha^{f,\sigma}\right]. \tag{D.7}$$

where the normalized error margin $\delta_\alpha^{f,\sigma}$ is scaled up by $\widehat{\sigma}(X_{new})$.

Split CP can be extended to quantile (and interval) base predictors $q(\cdot)$ via the nonconformity score:

$$R_i = \max\{\widehat{q}_{\alpha_{lo}}(X_i) - Y_i, Y_i - \widehat{q}_{1-\alpha_{hi}}(X_i)\}, \quad i = 1, \ldots, |D_{calibration}|. \tag{D.8}$$

This gives the *Conformalized Quantile Regression* (CQR) method [Romano et al., 2019], with PI:

$$\widehat{C}_\alpha(X_{new}) = \left[\widehat{q}_{\alpha_{lo}}(X_{new}) - \delta_\alpha^q\,,\, \widehat{q}_{1-\alpha_{hi}}(X_{new}) + \delta_\alpha^q\right], \tag{D.9}$$

where δ_α^q is a **correction** margin that guarantees finite-sample coverage for the quantile predictions. If $\delta_\alpha^q \approx 0$, we get a confirmation of its predictive marginal validity backed by theory.

For small datasets, the *jackknife+* [Barber et al., 2021] uses either a *Leave-One-Out* (LOO) or *K–fold* data schemes for better statistical efficiency, at the cost of fitting n_{train} and K models. When the base predictor is an ensemble learner using *bagging* [Breiman, 1996a], one can compute the nonconformity scores via the *Out-of-Bag* (OOB) "trick" [Breiman, 1996b] with negligible computational overhead[10]. Notably, the *Jackknife+–after–Bootstrap* [Kim et al., 2020] and the *Quantile Out–of–Bag* [Gupta et al., 2019] CP methods are, respectively, compatible with any point $f(\cdot)$ and interval $q(\cdot)$ predictor. For the above methods, the inference (Step 7) is modified to account for the multiple fitted predictors.

D.3.1.3 Conformal prediction for time series

As previously mentioned in the introduction, some recent publications have proposed extensions of CP to non-exchangeable data [Barber et al., 2022, Tibshirani et al., 2019, Xu and Xie, 2021]. For time series in particular, there are additional assumptions on the prediction problem to get close to the properties of CP. The method in [Chernozhukov et al., 2018] aims at recovering data exchangeability via permutations of sequences of data samples. Furthermore, Gibbs and Candès [2021] and Zaffran et al. [2022] worked on obtaining adaptive miscoverage probabilities α for online CP, while in [Stankeviciute et al., 2021] CP is applied to multiple time series assumed to

[10] When *bagging*, one fits B predictors on B bootstrap datasets S_b, sampled *with replacement* from D_{train}. Following a probabilistic fact (see citations), on average, about $\sim 30\%$ of the samples are left out of S_b, or *out-of-bag*, and we can compute the scores with out-of-sample data, approximating a *Leave-One-Out* (LOO) scheme.

be exchangeable. Finally, Diquigiovanni et al. [2021] worked with multivariate functional time series applied to demand prediction in the Italian gas market, and in [Wisniewski et al., 2020] we find a modified, empirically tested, version of split CP applied to financial data.

Among the most promising results is the *Ensemble Batch Prediction Intervals* (EnbPI) algorithm [Xu and Xie, 2021], a modification of the Jackknife+-after-Bootstrap that also uses OOB estimation of nonconformity scores (see Footnote 10). By doing so, EnbPI avoids overfitting without recourse to data-splitting, which improves the computational efficiency. Using the Out-of-Bag (OOB) trick [Breiman, 1996b], they compute the nonconformity scores and conformalize an ensemble algorithm with validity on time series data.

By construction, EnbPI yields constant-size intervals. We proposed [Mendil et al., 2022] and implemented a locally adaptive extension referred to as *Adaptive Ensemble batch Prediction Intervals* (aEnbPI), adding an ensemble estimation of the conditional *Mean Absolute Deviation* (MAD) $\widehat{\sigma}$ to the original algorithm. These OOB MAD estimators are obtained by aggregation of bootstrap models fitted on training data. In addition, the absolute residuals are replaced by a scaled version that serves to build adaptive PIs during inference. This is a straightforward application of LACP to EnbPI, yielding scaled nonconformity scores.

D.4. Recent developments in conformal prediction for sequential data

Stemming from the first results reviewed in 2021 and mentioned in Section D.3.1.3, this section extends the topic of CP for time series with some of the latest (as of late 2022) and most noteworthy papers published on CP for time series. Our experiments are restricted to the most promising for our UC.

D.4.1 Online sequential split conformal prediction (OSSCP)

The *Online Sequential Split Conformal Prediction* (OSSCP) algorithm, first introduced by Wisniewski et al. [2020] and later named, studied and used as baseline by Zaffran et al. [2022], is a straightforward extension of Split CP (see Section D.3.1.1) to an online prediction setting. At each time step, using a rolling time window, they split the training data into fit and calibration partitions without randomization: the proper training data precedes the calibration data. A depiction of this scheme is found in Figure D.2. The underlying idea is that using more recent samples as calibration can account for local changes, closer to the test point, while using a contiguous sequence of data for training can help building models which are more aware of patterns in the time series. Furthermore, using a *sequence* of samples instead of a random sample allows to also fit algorithms for time series such as ARIMA.

OSSCP can be used as a baseline to compare against specialized algorithms ACI, AgACI (see Section D.4.3) or *Non-Exchangeable Split Conformal Prediction* (NESCP) (Section D.4.5). Since time series data break the hypothesis of exchangeability, they lack the theoretical guarantees of CP. However, the method appears to **perform reasonably well** in practice. Furthermore, it can be shown that the fundamental theorem of CP, which relies on the exchangeability of calibration and test points $\{(X_i, Y_i)\}_{i=1}^{n+1}$, can be rewritten to rely only the exchangeabil-

ity of prediction errors (or generally nonconformity scores); for instance, take as the starting point the formulation given by Romano et al. [2019]. While time series data have by definition time-dependent pattern, this does not necessarily imply that the sequences of prediction errors $\{\varepsilon_t\}_{t=t_0}^{t_0+T}, \varepsilon_t = y_t - \widehat{y}_t$, have such patterns. The residuals of a correctly specified model for an ARIMA process are assumed to be i.i.d., for instance. While this observation does not solve the problem of exchangeability, it can shed light on why they can work well in some cases. A similar observation is made in the paper of EnbPI [Xu and Xie, 2021].

D.4.2 *Ensemble Batch Prediction Intervals* (EnbPI) and its variations

The EnbPI algorithm [Xu and Xie, 2021] (see Section D.3.1.3), already included in the report of batch 1 [Nabhan et al., 2022], has been modified and updated by the original authors and uploaded to arXiv in an extended format in Xu and Xie [2022].

Recall that the original EnbPI trains an ensemble of B predictors on B bootstrap samples[11] of the original training data, and does bootstrap aggregation [Breiman, 1996a] at inference. Following a simple probabilistic argument, having $B \gtrapprox 30$ should ensure that each sample is omitted at least in one of the bootstrap samples. Conformalization requires that the nonconformity scores are computed on holdout data, and this is possible thanks to the OOB trick [Breiman, 1996b]: let $f^{\phi}_{-t}(x_t)$, $t = 1, \ldots, T$, be the predictor that is the aggregation (via the function $\phi(\cdot)$) of all predictors which *did not* use the t-th sample in their training; then, the EnbPI (version 1) scores are computed as $\widehat{\varepsilon}^{\phi}_t = |y_t - f^{\phi}_{-t}(x_t)|$.

For the inference, let $f^{\phi}_{-(T+1)}(X_{T+1}) = (1-\alpha)$-th quantile of $\{f^{\phi}_{-i}(X_{T+1})\}_{i=1}^{T}$ be the point prediction for the unobserved Y_{T+1}. Then, the prediction interval is simply:

$$C_{\alpha}(X_{T+1}) = f^{\phi}_{-(T+1)}(X_{T+1}) \pm (1-\alpha)\text{-th quantile of } \{\widehat{\varepsilon}^{\phi}_t\}_{t=1}^{T}$$

Furthermore, they add a mechanism to update the residuals (hence the conformalizing quantile) to account to shifts in the distributions of the data, without needing to refit the predictors.

D.4.2.1 Ensemble Batch Prediction Intervals, version 2

Under a series of assumptions on the error process (errors being "well-behaved", their distribution being well approximated, and others), the authors show they can predict a valid and optimally short prediction interval, conditionally on $X_t = x_t$. In practice, from our experience, these special assumptions are rather hard to be met, and their theoretical guarantees must be taken with care. For instance, being able to achieve conditional coverage when the prediction errors are i.i.d. is not a ground-breaking result. In this case, the empirical CDF $\widehat{F}_t$ of $\{\varepsilon_t\}_t$ will accurately estimate the true F_t and return the α_{lo}-th and $(1-\alpha_{hi})$-th quantiles that assure coverage at the $(1-\alpha)$ level.

As for the technical modifications, version 2 of EnbPI now uses the raw prediction errors as nonconformity scores (no absolute value):

$$\widehat{\varepsilon}^{\phi}_t = y_t - f^{\phi}_{-t}(x_i) \tag{D.10}$$

[11] They do not take into account the ordering of the data when bootstrapping, uniformly at random, the samples.

They attain an optimally shortest interval, going through the estimation of a β parameter that shifts "to the right" or "to the left" the estimation of the empirical quantiles. This is possible because the scores in Equation D.10 keep their sign, unlike what is commonly done in CP (e.g. absolute deviation), which entails a loss of information on any asymmetry in the distribution of the prediction errors.

$$\begin{aligned} C_\alpha(X_{T+1}) &= \left[q_{lo}(\widehat{\beta}), q_{hi}(\widehat{\beta})\right] \\ &= \big[f^{\phi}_{T+1}(X_{T+1}) + (\widehat{\beta}) \text{ quantile of } \{\widehat{\varepsilon}^{\phi}_t\}_{t=1}^{T}, \qquad \text{(D.11)} \\ &\quad\; f^{\phi}_{T+1}(X_{T+1}) + (1-\alpha+\widehat{\beta}) \text{ quantile of } \{\widehat{\varepsilon}^{\phi}_t\}_{t=1}^{T}\big], \qquad \text{(D.12)} \end{aligned}$$

where:

$$\widehat{\beta} = \text{argmin}_{\beta\in[0,\alpha]}[q_{hi}(\beta) - q_{lo}(\beta)] \quad \text{(size of interval)}. \qquad \text{(D.13)}$$

Remark: for $\beta = 0$, the authors follow the convention of taking the 0-th empirical quantile of $\{\widehat{\varepsilon}^{\phi}_t\}_{t=1}^{T}$ to be $\min(\{\widehat{\varepsilon}^{\phi}_t\}_{t=1}^{T})$.

D.4.2.2 Ensemble Conformalized Quantile Regression (EnCQR)

Finally, we mention the work of Jensen et al. [2022], who merged the EnbPI algorithm with the Conformalized Quantile Regression of Romano et al. [2019]. Their algorithm, *Ensemble Conformalized Quantile Regression* (EnCQR), is a straightforward extension of EnbPI, useful when the variability of $Y|X$ is known or suspected to be not constant. They do not provide specific theoretical guarantee for their variant algorithm, although in practice we can expect it to work reasonably well given that both CQR and EnbPI seem to perform well in a variety of scenarios.

D.4.3 *Adaptive Conformal Inference* (ACI) and its variations

In this section we present the contents of the papers by Gibbs and Candès [2021], Gibbs and Candès [2022], originally framed for generic online prediction and Zaffran et al. [2022], focused on time series and their interactions with adaptive miscoverage rates. Furthermore, we can find some performance gains, for a reasonable additional computational cost, in the *Fully Adaptive Conformal Inference* (FACI) algorithm of Gibbs and Candès [2022], which includes the *Online Expert Aggregation on ACI* (AgACI) of Zaffran et al. [2022] as a special case.

D.4.3.1 Adaptive conformal inference

The *Adaptive Conformal Inference* (ACI) [Gibbs and Candès, 2021] algorithm requires the baseline conformal predictor to be parameterized by an adaptive miscoverage rate $\alpha_t \in (0,1)$, which can change at each time step and hence "adapt" to distribution shifts of the data. Setting $\alpha_1 = \alpha$ and $\gamma > 0$, for $t \geq 1$ the update is defined as:

$$\alpha_{t+1} = \alpha_t + \gamma(\alpha - \text{err}_t), \qquad \text{(D.14)}$$

where $\alpha \in (0,1)$ is the target miscoverage probability set by the user, α_t and $\text{err}_t = \mathbb{1}\{Y_t \notin \widehat{C}_{\alpha_t}(X_t)\}$ are respectively the adaptive miscoverage rate and the empirical error of iteration t. The parameter $\gamma > 0$ determines how "reactive" the algorithm is, when updating α_{t+1} after correct ($\text{err}_t = 0$) and erroneous ($\text{err}_t = 1$) predictions. In their experiments, Gibbs and Candès set $\gamma = 0.005$, as it was found to be empirically satisfactory. The authors also experiment with the following weighted variation:

$$\alpha_{t+1} = \alpha_t + \gamma \left(\alpha - \sum_{s=1}^{t} w_s \text{err}_s \right) \tag{D.15}$$

where $\{w_s\}_{1 \leq s \leq t}, w_s \in [0,1]$, is a sequence of increasing weights such that $\sum_{s=1}^{t} w_s = 1$. They find that they yield "almost identical results" although the sequences of $\{\alpha_t\}_{t=1|}^{T}$ stemming from Equation D.15 are smoother.

D.4.3.2 *Online Expert Aggregation on ACI* (AgACI)

The paper by Zaffran et al. [2022] studies the behaviour of ACI by Gibbs and Candès [2021] on time series data. They also introduce and test their own version of the algorithm, *Online Expert Aggregation on ACI* (AgACI), which works by carrying out K updates in parallel, yielding a set $\{\alpha_t^k\}_{k=1}^{K}$ adaptive miscoverage rates at each t-th iteration, and aggregating their predictions. Casting the problem as one of online learning [Cesa-Bianchi and Lugosi, 2006, Hazan et al., 2016], each update is guided by a different value γ_k, referred to as "expert' in the literature, and the resulting predictions $\{\widehat{C}_t^k(X_t) = \left[l_t^k, u_t^k\right]\}_{k=1}^{K}$ are aggregated using the Bernstein online aggregation of Wintenberger [2017].

They show how different values of γ have a direct influence over the empirical coverage of the algorithm. Among other things, they simulate data $Y_t = f(X_t) + \varepsilon_t$, with noise from an (autoregressive moving average) ARMA(1,1) process of parameters ϕ and θ, that is, $\varepsilon_{t+1} = \phi\varepsilon_t + \xi_{t+1} + \theta\xi_t$, where ξ is a white noise known as *innovation* in the time series literature. They give an empirical example of how CP methods, applied to non-exchangeable data, can attain good coverage, $\approx (1-\alpha)100\%$, for values of $\theta = \phi \leq 0.9$.

On a practical standpoint, this new empirical study and the previous results from the literature [Wisniewski et al., 2020, Mendil et al., 2022], strengthen our belief that CP is a good starting point also when applied to non-exchangeable data. Of course, in this later case, one can easily improve the performance of CP using the special algorithms introduced in this chapter.

D.4.3.3 *Fully Adaptive Conformal Inference* (FACI)

The *Fully Adaptive Conformal Inference* (FACI) algorithm [Gibbs and Candès, 2022], which includes AgACI [Zaffran et al., 2022] as a special case, requires a pool of γ candidate values γ_k, $k = 1, \ldots, K$. The idea is to carry on K updates in parallel, at each iteration, choosing the γ_k that yields the best prediction according to some criterion. FACI is not computationally prohibitive if the baseline method (*e.g.* Split CP) can produce K prediction intervals, one for each γ_k, with negligible computational overhead.

With abuse of notation, we denote with $C_t(\beta)$ a prediction set built using the probability of order $1-\beta$ to compute the empirical quantile (see Section D.3.1.1); unlike with proper Split CP (see Step 6 of Section D.3.1.1) and derived methods, FACI requires no adjustment in the empirical quantile formula because its validity relies on a different mathematical argument (notably, FACI is not valid in finite samples).

For the sake of this, let us take a regression example. An acceptable nonconformity score could be: $S_t(X_t, y; (X_i, Y_i)_{i=1}^{t-1}) = |y - \widehat{f}_t(X_t)|$. Here,$(X_i, Y_i)_{i=1}^{t-1}$ is the recorded data up to time $t-1$, used to train or update the base predictor $\widehat{f}_t$. The RCI prediction interval is then defined implicitly as:

$$C_t(\beta) = \left\{ y : S_t^y \leq \text{Quantile}\big(1-\beta, \frac{1}{t}\sum_{i=1}^{t} \delta_{S_i^y}\big) \right\}. \tag{D.16}$$

Also, let us define the following quantile loss, also known as *pinball* loss:

$$\ell(\beta_t, \theta) := \alpha(\beta_t - \theta) - \min\{0, \beta_t - \theta\}, \tag{D.17}$$

where

$$\beta_t := \sup\{\beta : Y_t \in C_t(\beta)\} \tag{D.18}$$

Along the lines of the argument of the authors (convexity of loss function ℓ), we have that the following update formula is equivalent to Equation D.14:

$$\alpha_t := \alpha_{t-1} - \gamma \nabla_\theta \ell(\beta_t, \theta) \tag{D.19}$$

We set a value α_t^k for each $\gamma_k, k = 1, \ldots, K$, which is updated online following Equation D.19. At inference, the value α_t to be used for to build the prediction interval is selected according to the historical performance of the values $\{\gamma_k\}$, (we leave the details to the original paper [Gibbs and Candès, 2022]).

D.4.3.4 Rolling conformal inference

Similarly to the ACI methods (Section D.4.3), Feldman et al. [2022] propose a new method for online prediction, called *Rolling Conformal Inference* (RCI), which aims at being adaptive with respect to possible (time-varying) shifts in the $\mathbb{P}_{XY}$ distribution of the data. Unlike with ACI, instead of updating a parameter that influences the size of the prediction intervals, the interval size is directly parameterized by a calibration parameter θ_t. The idea is that larger (respectively smaller) values of θ will yield larger (respectively smaller) prediction sets, thus forcing the intervals to cover, on the long term, the true values of Y with a nominal level $1-\alpha$.

This is framed as an online prediction task, hence for each time step t we assume the following phases: first, the test feature vector X_t is revealed, then we build a prediction interval around the predicted $\widehat{Y}_t$, then the true observation Y_t is revealed and finally we update the parameter θ_t and the predictor $f(\cdot)$ with this new data. The prediction interval at time t is determined by:

$$\widehat{C}_t^{RCI}(X_t) = f(X_t, \theta_t, \mu_t), \tag{D.20}$$

where the interval predictor $f(\cdot)$ takes as inputs: the current test feature vector X_t, the calibration parameter θ_t and the model $\mu_t(\cdot)$, predicting the target $Y|X$, fitted on data points $\{X_t, Y_t\}_{i=1}^{t-1}$. When the true observation Y_t is revealed, the calibration parameter is *tuned* according to:

$$\theta_{t+1} = \theta_t + \gamma(\text{err}_t - \alpha), \tag{D.21}$$

with $\gamma > 0$ and $\text{err}_t = \mathbb{1}\{Y_t \notin \widehat{C}_{\alpha_t}^{RCI}(X_t)\}$. Remark that this is slightly different from Equation D.14 of ACI, where the update is $\alpha - \text{err}_t$.

Finally, we get μ_{t+1} by updating the previous model μ_t with (X_y, Y_t). This can be done seamlessly, for instance, with online learning models or simply by refitting μ_{t+1} on the whole data sequence $\{X_t, Y_t\}_{i=1}^{t}$. As demonstrated in Theorem 1 of Feldman et al. [2022], their algorithm is **valid** in the following sense:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{1}\{Y_t \notin \widehat{C}_{\alpha_t}^{RCI}(X_t)\} = 1 - \alpha \tag{D.22}$$

Like in conformal inference (Section D.3), this method is **distribution-free** and **model-agnostic**: the underlying distribution need not be known, nor we need assumptions on $\mu(\cdot)$. However, the coverage is guaranteed only **asymptotically**.

As to the **actual construction** of the conformal prediction interval using θ, the authors propose multiple solutions. For the case of quantile regression, let $\mu(X_t, \frac{\alpha}{2})$ and $\mu(X_t, 1-\frac{\alpha}{2})$ be lower and upper quantile predictors (quantile regression). The RCI interval is then:

$$f(X_t, \theta_t, \mu_t) = \left[\mu_t(X_t, \frac{\alpha}{2}) - \phi(\theta_t), \mu_t(X_t, 1-\frac{\alpha}{2}) + \phi(\theta_t)\right] \tag{D.23}$$

This interval predictor looks very similar to what prescribed by CQR [Romano et al., 2019, see Equation D.9], except that instead of computing the quantile δ_α^q of a set of holdout residuals (nonconformity scores), we use an arbitrary *stretching function* $\phi(\theta_t)$. A trivial example could be $\phi(\theta) = \theta$: since θ is not bounded, it could directly represent the adjustment (error margin). However, given the formula in Equation D.21, the update could happen too slowly. A better alternative is to use the **exponential stretching**:

$$\phi(\theta) = \begin{cases} e^{\theta} - 1, & \theta > 0 \\ -e^{-\theta} - 1, & \theta \leq 0 \end{cases} \tag{D.24}$$

With this function, for $\theta \approx 0$, the calibration would be moderate while for bigger values the inflation of the prediction interval would grow exponentially. Intuitively, if the PIs can grow rapidly, we start capturing a good fraction of true values "sooner" and the adjustment can have effect in fewer iterations. The authors also adapt this method to tasks other than regression, such as classification and multi-output regression.

D.4.4 Conformal Prediction Under Covariate Shift

The work of Tibshirani et al. [2019] is an extension of the conformal prediction in case of non-exchangeable data. More specifically, it leverages the concept of "weighted exchangeability" to handle covariate shift.

According to Storkey [2009], covariate shift occurs when the data is generated according to a model $P(Y/X)P(X)$ and where the distribution $P(X)$ changes between training and test scenarios.

$$\textbf{Train data: } (X_i, Y_i) \sim P = P_X \times P_{Y/X}, \quad i = 1, \ldots, n$$

$$\textbf{Test data: } \quad (X_{n+1}, Y_{n+1}) \sim \widetilde{P} = \widetilde{P}_X \times P_{Y/X}$$

Notice that the conditional distribution of Y/X is assumed to be the same for both the training and test data (there is no concept shift).

D.4.4.1 Weighted Split Conformal

In the assumption of covariate shift, the calibration and test dataset are no longer exchangeable. The procedure of conformal prediction remains similar except that the nonconformity scores are weighted by a probability proportional to the density ratio:

$$w(X_i) = \frac{f_{\text{test}}(X_i)}{f_{\text{calib}}(X_i)}, \tag{D.25}$$

where f_{test} and f_{calib} are the marginal probability density functions of X under the test and calibration distributions, respectively.

Given a calibration set $\mathscr{D}_{calib} = \{(X_i, Y_i)\}_{i=1,\cdots,n}$ and a test point (X_{n+1}, Y_{n+1}), the weighted residuals distribution is as follows:

$$f_R = \sum_{i=1}^{n} \tilde{w}_i \cdot \delta_{R_i} + \tilde{w}_{n+1} \cdot \delta_{+\infty}, \tag{D.26}$$

where $R_i = |Y_i - \widehat{\mu}(X_i)|$ for the pre-trained model $\widehat{\mu}$,

$$\tilde{w}_i = \frac{w(X_i)}{w(X_1) + \cdots + w(X_n) + w(X_{n+1})}, \text{ and } \tilde{w}_{n+1} = \frac{w(X_{n+1})}{w(X_1) + \cdots + w(X_n) + w(X_{n+1})}.$$

Therefore, the interval prediction for the new point (X_{n+1}, Y_{n+1}) is:

$$\begin{aligned} \widehat{C_n}(X_{n+1}) = [\widehat{\mu}(X_{n+1}) - Q_{1-\alpha}(\sum_{i=1}^{n} \tilde{w}_i \cdot \delta_{R_i} + \tilde{w}_{n+1} \cdot \delta_{+\infty}), \\ \widehat{\mu}(X_{n+1}) + Q_{1-\alpha}(\sum_{i=1}^{n} \tilde{w}_i \cdot \delta_{R_i} + \tilde{w}_{n+1} \cdot \delta_{+\infty})], \end{aligned} \tag{D.27}$$

where that $Q_{1-\alpha}(.)$ is the $(1-\alpha)$th empirical quantile.

In this setup, data are "weighted exchangeable" and the conformal prediction has a valid coverage.

D.4.4.2 Weighted exchangeability

Definition [Tibshirani et al. 2019]: Random variables $V_1, \ldots, V_n$ are said to be weighted exchangeable with weights functions $w_1, \ldots, w_n$ if their joint probability density function f can be factorized as

$$f(v_1, \ldots, v_n) = \prod_{i=1}^{n} w_i(v_i) \cdot g(v_1, \ldots, v_n),$$

where g is any function that does not depend on the ordering of its inputs, i.e., $g(v_{\sigma(1)}, \ldots, v_{\sigma(n)}) = g(v_1, \ldots, v_n)$ for any permutation σ of $1, \ldots, n$.

Lemma [Tibshirani et al., 2019]: Let $Z_i \sim P_i$, $i = 1, \ldots, n$ be independent draws, where each $P_{i \geq 2}$ is absolutely continuous with respect to P_1. Then $Z_1, \ldots, Z_n$ are weighted exchangeable, with weight functions $w_1 : x \to 1$ and $w_i = dP_i/dP_1$ for $i \geq 2$.

In the special case of the weighted split conformal, data are weighted exchangeable by construction of the weights in Equation (D.25). From **Lemma 3** in Tibshirani et al. [2019], we conclude that

$$\mathbb{P}\{R_{n+1} \leq Q_{1-\alpha}(R_w)\} \geq 1 - \alpha. \tag{D.28}$$

As a consequence, the prediction interval of the weighted split conformal is marginally valid.

D.4.4.3 Weight Estimation

In case we do not know the true probability density functions f_{test} and f_{calib}, we want to estimate their ratio from the available data. There are several methods to approach the density ratio estimation; Tibshirani et al. chose the probabilistic classification method.

Let $X_1, \ldots, X_n$ be the samples in $\mathscr{D}_{calib}$ and $X_{n+1}, \ldots, X_{n+m}$ be the samples in $\mathscr{D}_{test}$. We consider the binary classification problem on (X_i, C_i) where $X_i \in \mathscr{D} = \mathscr{D}_{calib} \cup \mathscr{D}_{test}$ and:

$$C_i = \begin{cases} 0 & \text{if } X_i \in \mathscr{D}_{calib} \\ 1 & \text{if } X_i \in \mathscr{D}_{test} \end{cases}, \tag{D.29}$$

We note $\widehat{p}(x)$ the estimate of $\mathbb{P}(C = 1 | X = x)$ obtained by fitting a classifier to the data $\{(X_i, C_i)\}_{i=1,\ldots,n+m}$. We have:

$$
\begin{aligned}
w(x) &= \frac{f_{\text{test}}(x)}{f_{\text{calib}}(x)} \\
&\sim \frac{\mathbb{P}(X=x|C=1)}{\mathbb{P}(X=x|C=0)} \\
&= \frac{\mathbb{P}(C=1|X=x)\cdot\mathbb{P}(X=x)}{\mathbb{P}(C=1)} \cdot \frac{\mathbb{P}(C=0)}{\mathbb{P}(C=0|X=x)\cdot\mathbb{P}(X=x)} \\
&= \frac{\mathbb{P}(C=0)}{\mathbb{P}(C=1)} \cdot \frac{\mathbb{P}(C=1|X=x)}{\mathbb{P}(C=0|X=x)} \approx \frac{\mathbb{P}(C=0)}{\mathbb{P}(C=1)} \cdot \frac{\widehat{p}(x)}{1-\widehat{p}(x)}.
\end{aligned}
\tag{D.30}
$$

Since the weights are normalized (see Equation D.26), it is sufficient to approximate the density ratio w proportionally to a constant. Therefore, the density ratio $\widehat{w}$ is:

$$
\widehat{w}(x) = \frac{\widehat{p}(x)}{1-\widehat{p}(x)}. \tag{D.31}
$$

D.4.5 Non-exchangeable Conformal Prediction

The work of Barber et al. [2022] proposes an extension of the split conformal, full conformal and jackknife+ methods in case the data points are non-exchangeable. Specifically, the potential coverage gap due to this non-exhangeability is **explicitly upper-bounded** with respect to the total variation distance between the in-sample and out-of-sample data distributions and weights w expressing the likelihood of the data to come from the same distribution.

For non-exchangeable conformal methods, *Barber et al.* introduce weights $w_1,\dots,w_n \in [0,1]$ with the intuition that a higher weight w_i should be assigned to a data point $Z_i = (X_i, Y_i)$ that is "trusted", i.e., that is believed to come from (nearly) the same distribution as the test point Z_{n+1}. The weights w_i are assumed to be fixed (contrarily to Tibshirani et al. [2019] that construct weights from the data).

Particularly, the split conformal is modified to use weighted quantiles, with weights given by the w_i rather than the orginal difinitions where all data points are implicitly given equal weights. The prediction interval is given by:

$$
\widehat{C_\alpha}(X_{n+1}) = \left[\widehat{\mu}(X_{n+1}) \pm Q_{1-\alpha}\left(\sum_{i=1}^{n} \tilde{w}_i \delta_{R_i} + \tilde{w}_{n+1}\cdot\delta_{+\infty}\right)\right], \tag{D.32}
$$

where $R_i = |Y_i - \widehat{\mu}(X_i)|$ for the pre-trained model $\widehat{\mu}$ and

$$
\tilde{w}_i = \frac{w_i}{w_1+\cdots+w_n+1},\ i=1,\dots,n \text{ and } \tilde{w}_{n+1} = \frac{1}{w_1+\cdots+w_n+1}. \tag{D.33}
$$

Let $Z = (Z_1,\dots,Z_{n+1})$ denote the full data set, where $Z_i = (X_i, Y_i)$ composed of n calibration points $(Z_1,\dots,Z_n)$ and one test point Z_{n+1}. Let $Z^i = (Z_1, Z_{i-1}, Z_{n+1}, Z_{i+1}, Z_n, Z_i)$ denote the same data set after swapping the test point Z_{n+1} with the i-th calibration point Z_i $(i = 1,\dots,n)$.

Theorem [Barber et al. 2022]: Let $\widehat{\mu}$ be any pre-fitted model. The non-exchangeable split conformal method defined in (D.32) satisfies:

$$\mathbb{P}(Y_{n+1} \in \widehat{C}_\alpha(X_{n+1})) \geq 1 - \alpha - \underbrace{\sum_{i=1}^{n} \tilde{w}_i \cdot d_{TV}(R(Z), R(Z^i))}_{\text{Coverage Gap}}, \tag{D.34}$$

where $R(Z) \in \mathbb{R}^{n+1}$ (resp. $R(Z^i) \in \mathbb{R}^{n+1}$) is the residual vector built from the inferences with $\widehat{\mu}$ on Z (resp. Z^i).

The authors explain the intuition for choosing the weights w_i: higher weights should be associated with calibration data points (X_i, Y_i) that are likely to be drawn from a "similar" distribution as the test point (X_{n+1}, Y_{n+1}). Conversely, lower weights are assigned to data points that are less reliable. The theorem (D.34) suggests that the coverage gap can be decreased by picking small weights w_i. However, this comes with a downside: the smaller the weights, the larger the prediction intervals are. In the extreme case where $w_1 = w_2 = \cdots = w_n = 0$, the coverage gap is zero but the prediction interval covers all real values $\widehat{C}_\alpha(X_{n+1}) = \mathbb{R}$, which is completely uninformative.

Empirically, for time series, we recommend simple choices such as weights that decay exponentially over time:

$$w_i = \rho^{n+1-i},$$

such that n is the size of calibration dataset, i is a time index and $\rho \in (0,1)$ is the decay parameter. Notice the weights associated to more recent samples are larger: $i_1 \geq i_2 \implies w_{i_1} \geq w_{i_2}$.

D.5. Experiments

D.5.1 Use Case: Air Liquide Demand Forecasting[12].

We tackle a use case of industrial gas demand forecasting. An anonymized dataset was built by Air Liquide, a world leader in gases, technologies and services for industry, to be representative of the industrial process. In the case of this work, it takes part in the overall characterization of the AI algorithm, by tackling the uncertainty associated with an AI algorithm, and finding well-defined metrics and quantification approaches with high guarantee.

D.5.1.1 The Industrial Use Case

With several sources of production and multiple customers of industrial gases across France, it is the company's responsibility to guarantee the availability of supply, i.e. it should know when to deliver the right product to all its customers on time.

[12]The opening of Section D.5.1, Section D.5.1.1 and D.5.1.2 are (lightly edited) versions of Section 1 in Mendil et al. [2022]

By taking into account geographical and logistical factors, managing the customers' products storage has an effect on the whole supply chain, from production to distribution. Therefore, the dispatchers, who are in charge of delivering the products, estimate the future trends of customers demands in advance based on their years of experience in the field. They have to take into account daily logistical constraints, e.g., limited transport resources, limited product availability, and accessibility issues.

In order to help the dispatchers in their estimations, multiple forecasting algorithms have been proposed to predict the needs in production as precisely as possible in the short term, and compare them with the dispatchers' estimations. Nonetheless, despite achieving remarkable performance, the comparison between the predictions and the estimations yields an uncertainty that is currently not measured. Such uncertainty can have a critical impact on production: if we produce more than intended, we face energy and product losses; conversely, if we produce less, we risk drying out the customers and failing to deliver the right amounts on time.

D.5.1.2 Challenges and Objectives

In light of this industrial context, even the best forecast model can suffer from uncertainty between the forecast predictions and the dispatchers estimations. The focus is then shifted toward **quantifying** the uncertainty associated with the forecast predictions. Since this forecasting use case is a regression problem, one effective way would be by building prediction intervals via *Uncertainty Quantification* (UQ), which consist of upper and lower bounds that contain the forecast predictions with high probability. Hence, the dispatchers would have at their disposal minimum and maximum values that numerically quantify operational uncertainties.

Nonetheless, operational constraints have been expressed regarding the desired results. Obviously, these intervals should have a high coverage probability to be used in production with great trust, while also being as "narrow" as possible to be informative for the dispatchers. To approximate more effectively their predictions, dispatchers need a sort of statistical safety net in their decision-making process, that will increase the robustness of these operations, boost the trust in the forecast algorithm, and optimize the production and distribution of the products.

D.5.1.3 Structure of the regression problem and choice of validity

The dataset of our UC was built aggregating 29 time series (referred to as "subseries" in the text), recorded weekly; at any time step we have 29 entries. To help comprehension of later sections, here is a **simplified** version of the dataset as provided by Air Liquide. Let $t = 1, ..., T$ be the time index, $C = \{c_1, c_2, \ldots, c_{29}\}$ be a categorical variable encoding the subseries and X_A one of the features.

We work with the UC data as structured in the simplified Table D.1, although this could be re-written as in Table D.2, typically found in multi-output regression problems. This difference hints at a challenge behind the design of conformalization and assessing coverage: do we *need* to have the **whole target vector** to be covered at the $1-\alpha$ level? Or do we rather aim for an average coverage? That is, we must choose between the guarantee of $\mathbb{P}\{[Y_{T+1}^{c_1} \cdots Y_{T+1}^{c_{29}}] \in C_\alpha(X_{T+1})\} \geq 1-\alpha$, and $\mathbb{P}\{Y_{T+1}^{c_k} \in C_\alpha(X_{T+1})\} \geq 1-\alpha$, for $k \in \{1, \ldots, 29\}$. The choice depends on the operational need of the user. A naive solution for the first would be to opt for a an adjustment

t	Y	C	X_A
1	y_1^1	c_1	.
1	y_1^2	c_2	.
		⋮	
1	y_1^{29}	c_{29}	.
2	y_2^1	c_1	.
2	y_2^2	c_2	.
		⋮	
2	y_2^{29}	c_{29}	.
...			
T	y_T^1	c_1	.
T	y_T^2	c_2	.
		⋮	
T	y_T^{29}	c_{29}	.

Table D.1: A simplified rendition of our UC's dataset.

t	Y^{c_1}	Y^{c_2}	...	Y^{c_29}	$X_A^{c_1}$	$X_A^{c_2}$	...	$X_A^{c_{29}}$
1	$y_1^{c_1}$	y_1^2	...	y_1^{29}	.	.	...	.
2	$y_2^{c_1}$	y_2^2	...	y_2^{29}	.	.	...	.
⋮								
T	$y_T^{c_1}$	y_T^2	...	y_T^{29}	.	.	...	.

Table D.2: Example of multi-output dataset structure applied to our UC.

of the miscoverage rate *á la* Bonferroni: $\tilde{\alpha} = \frac{\alpha}{K}$, where K is the dimension of the target vector. This entails selecting a "more extreme" conformalizing quantile (see Step 6 in Section D.3.1.1) resulting in the inflation of the size of the prediction intervals (or rather *prediction regions*, in this multi-dimensional case). The reader can find an application of this in de Grancey et al. [2022], for the conformalization of object detection: for their UC (pedestrian detection), it is critical that the bounding box covers the entire object at the nominal coverage level $1-\alpha$. This can be attained, for instance, by controlling each one of the four sides of the bounding box, yielding $\tilde{\alpha} = \frac{\alpha}{4}$.

The previous solution is not the only way to approach this multidimensional problem. An alternative could be to formalize our problem following the definitions of coverage found in the recent paper by Lin et al. [2022]: the authors propose a notion a **longitudinal validity** and **cross-sectional validity** when working with multiple time series at once.

In Figure D.1, we see a simplified depiction of the two coverages: each series corresponds to a patient being followed up in a longitudinal study, the purpose is to be right $100\,(1-\alpha)\%$ of the time, for $100\,(1-\alpha)\%$ of the patients, at any time. As stated above, this choice is arbitrary

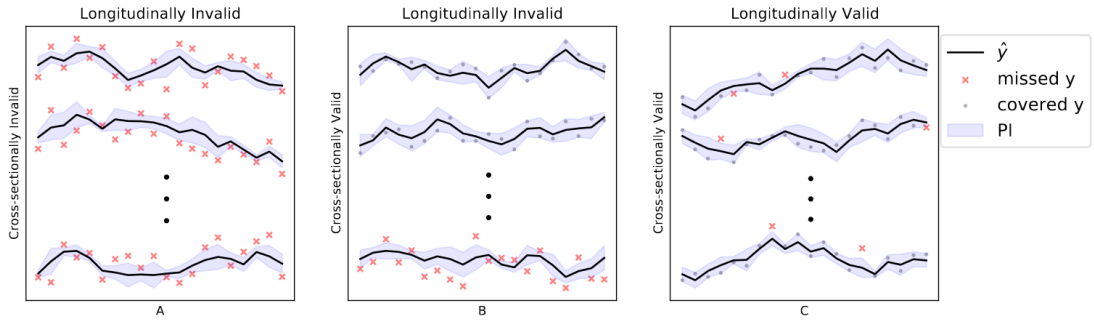


Figure D.1: Cross-sectional coverage is on the vertical axis, longitudinal is on the horizontal axis. Adapted from Figure 1 in Lin et al. [2022] (published under license: CC BY 4.0).

and depends on the user's needs.

This type of coverage is, informally, what we aimed for in Section D.6.2. Along the lines of Lin et al. [2022], we would want to observe that for each subseries the nominal coverage $1-\alpha$ is attained. There, we train our predictor on the data formatted as in Table D.1 and conformalize the series separately, computing 29 sets of nonconformity scores by conditioning as: $Y_t|C=1, Y_t|C=2, \ldots, Y_t|C=29$, as done also in Romano et al. [2020]. In practice, both using adaptive methods such as CQR and conditioning the conformalization by the category C seems to work well.

D.5.2 Updated experiment procedure

Following the scheme introduced in Section D.4.1, we work with OSSCP, an online version of inductive conformal prediction, depicted in Figure D.2. Initially, N data points are available, which we split sequentially into a fit subset $\mathscr{D}_{fit} = \{(X_1,Y_1),\ldots,(X_K,Y_K)\}$ and a calibration subset $\mathscr{D}_{calib} = \{(X_{K+1},Y_{K+1}),\ldots,(X_N,Y_N)\}$. As argued in Zaffran et al. [2022], not randomizing the samples aims at excluding future observations from $\mathscr{D}_{fit}$ and prevent over-fitting. Otherwise, such data leakage may lead to an under-estimation of the errors on $\mathscr{D}_{calib}$. For a new point (X_{N+1},Y_{n+1}), the prediction interval is estimated by $\hat{C}_\alpha(X_{N+1})$. At time step $t=N+2$, Y_{n+1} is already observed and available to be used in the conformalization process. The newest point enables to update the fitting and calibration subsets by shifting the time index by one: $\mathscr{D}_{fit} = \{(X_2,Y_2),\ldots,(X_{K+1},Y_{K+1})\}$ and $\mathscr{D}_{calib} = \{(X_{K+2},Y_{K+2}),\ldots,(X_{N+1},Y_{N+1})\}$. Then, we retrain/recalibrate the model using the new split and estimate the PI $\hat{C}_\alpha(X_{N+2})$ and so on. Note that we can consider a batch size $s>1$ for the update of the underlying regression model and nonconformity scores after s new observations.

D.5.3 New metrics

Comparatively to the usual metrics presented in Section D.2.3, we need specific adaptations to assess the robustness of the SoTa methods against changes in the data distribution with time.

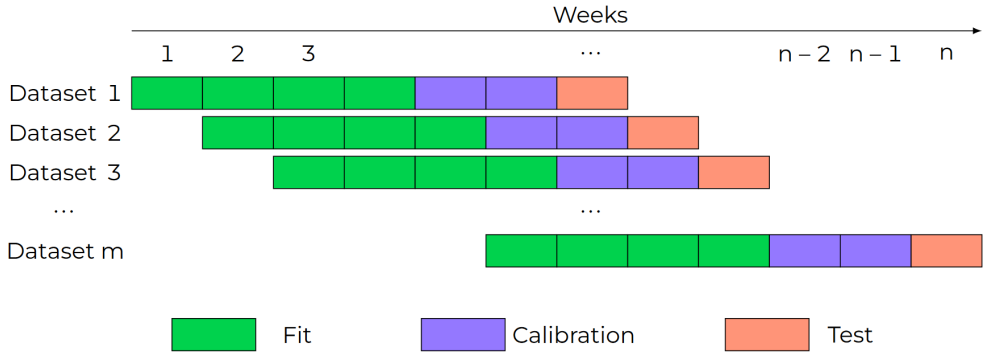


Figure D.2: The sequential split scheme used in the experiments. The sequence of n weeks in the original dataset is split into m datasets. The number of fit weeks (in green) and calibration weeks (in purple) can be tuned by the users.

Therefore, in addition to computing the marginal coverage probability (PICP) and the average PI width (PINAW) over the whole time horizon H, we will evaluate their evolution over smaller intervals (or windows). This helps us analyze and visualize the impact of data shifts occurring at certain times on the PI's validity and efficiency.

Let $\mathscr{D}_{test} = \{(X_{N+1}, Y_{N+1}), ..., (X_{N+H}, Y_{N+H})\}$ be a test set composed of H data points (revealed in a stream fashion). Following the procedure described in Section D.5.2, a PI $\hat{C}_\alpha(X_t) = [\hat{Y}_t^L, \hat{Y}_t^U]$ is estimated for each time step $t = N+1, \ldots, N+T$. We consider a constant time window of size h and define the moving average of the PICP for time window i as:

$$\text{MA-PICP}_i = \frac{1}{h} \sum_{k=i}^{i+h} \mathbb{1}\{\widehat{Y}_k^L \leq Y_k \leq \widehat{Y}_k^U\} \in [0,1]. \tag{D.35}$$

Similarly, the moving average PINAW for time window i is defined as follows:

$$\text{MA-PINAW}_i = \frac{1}{hR} \sum_{k=i}^{i+h} (\widehat{Y}_k^U - \widehat{Y}_k^L), \tag{D.36}$$

where $R = Y_{max} - Y_{min}$ is a normalization factor.

D.6. Results

In Section D.5.1.3 we mentioned the different kinds of validity admissible for our problem. In what follows, we first report the validity averaging the coverage among the 29 subseries at each time step, and then study how to attain the valid coverage of $1-\alpha$ for each individual case.

We also recall that the **empirical coverage** reported in this Section is a **random variable**, since its value depends on the sampling of the test data. Even for a textbook example built by simulation, we must expect to have some fluctuations around the nominal coverage level of $1-\alpha$. Angelopoulos and Bates [2021] gives a clear example of this in their introduction to CP.

Method	γ	Empirical Coverage ($\widehat{\sigma}$)		Average Width ($\widehat{\sigma}$)	
OSSCP	NA	0.901	(0.073)	1.017	(0.187)
OSCQR	NA	0.896	(0.082)	0.885	(0.457)
NESCP	0.999	0.898	(0.073)	1.003	(0.186)
NESCP	0.990	0.900	(0.075)	1.035	(0.231)
NESCP	0.980	0.926	(0.067)	1.228	(0.318)
NESCP	0.970	0.944	(0.058)	1.429	(0.382)

Table D.3: Experimental data, with nominal coverage of $(1-\alpha) = 0.9$: metrics averaged over 300 runs (standard deviation given in parentheses), following the scheme depicted in Figure D.2. For each run, we took 20 weeks of fit, 5 weeks of calibration and one week of test.

D.6.1 Comparison of methods, averaging coverage in the subseries

In Table D.3 and in Figure D.3, at each inferential step, we average the coverage over the 29 subseries and over the 300 weeks of test data. That is, for each test time $t \in [1,\dots,300]$, we compute 29 prediction intervals and count how many times we captured the true value of $Y_t^c, c = 1,\dots,29$. For instance, if we capture 23 out of 29 values, we say that at time t we have coverage of about 79%. The values we report are the average coverage (as in the PICP of Equation D.2) and the average interval size (the average width defined in Equation D.3) over 300 test steps and over 29 subseries. For instance, saying that the method OSSCP in Table D.3 attained a coverage of 0.901 means that over the 300×29 intervals built, 90.1% contained the true value. The same can be said for the average width of the intervals.

In Figure D.3 we see that all methods attained the threshold of 90% coverage within a few decimal points. NESCP using $\gamma = 0.98$ and $\gamma = 0.97$ yielded much bigger intervals and higher coverage: this is a results of the algorithm, which ensures better coverage by being more conservative (larger intervals).

The empirical results obtained exceeded our expectations, since OSSCP and OSCQR do not have any special theoretical guarantee for the case of time series data, just a healthy dose of common sense. An explanation could be that the prediction errors are well-behaved and do not exhibit egregious patterns: their behavior could be close to be exchangeable or even i.i.d., that is, we could have (unknowingly) that at every inferential step the value of the error does not depend on what observed or predicted at the previous step. This is an unlikely scenario in practice, and we suppose that the truth could be somewhere in the middle: within our UC, the prediction errors turned out to be well-behaved *enough* to grant us good empirical results. *Ça va sans dire*, we cannot expect this to hold in the general case and only an empirical, experimental assessment of various methods can help the user in choosing the right method.

If we leave the land of averages and enter the details of coverage for each subseries, we get a slightly different picture: some subseries are overcovered and others are undercovered. For this reason, we now study the performance of CP when conditioning the conformalization on the subseries values.

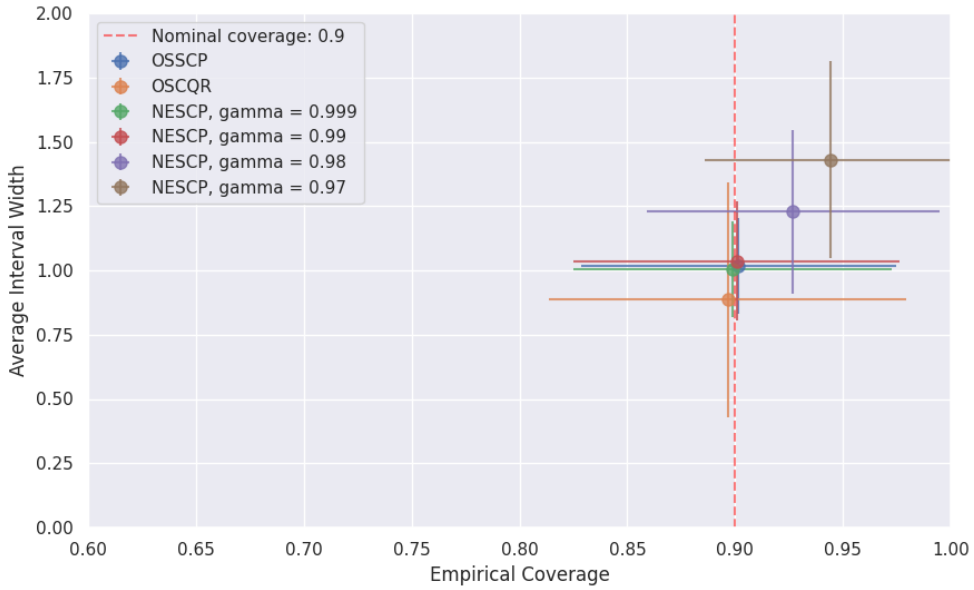


Figure D.3: Comparison of multiple CP methods. Empirical coverage vs Average width of prediction intervals, $\pm$ standard deviation.

D.6.2 Coverage for each subseries in the data

Air Liquide's dataset is composed of 29 time series, each considered as a category. The previous nonconformity scores and PIs were computed regardless of these categories. But the idea of group-conditional CP explored by [Romano et al., 2020] is interesting to fine-tune the calibration for each of the 29 times series individually. The incentive of such targeted calibration is to reach marginal validity for each individual time series, which is not theoretically guaranteed when the conformalization is applied indistinctively (see Table D.4). The regression model is, however, still trained across all the categories. The reason is there are underlying dependencies between the time series (confirmed by domain knowledge) that drastically improve the accuracy during inference. Nothing alarming here as CP is model-agnostic.

The experimental procedure is similar to the one described in D.5.2, the only difference being the use of 29 calibration datasets instead of one, where $\mathscr{D}^i_{calib}$ is related to time series i and used to infer the PI through $\hat{C}^i_\alpha$.

Table D.4 shows the performance of the PIs obtained for each time series' category with OSSCP (baseline) and NESCP in terms of marginal coverage and average width. The α is set to 10%, the target coverage is thereby 90%. Overall, the coverage with OSSCP is nearly valid. Rows highlighted in red correspond to significant coverage deviations from the target when OSSCP is used. The deviations are reduced with NESCP in exchange for an increase of PIs' widths. More specifically, we can track in Figure D.4 the evolution with time of the PI's validity and efficiency for four time series highlighted in red. The PI's coverage and width are averaged

Category	Empirical Marginal Coverage		Empirical Average Width	
	OSSCP	***NESCP***	***OSSCP***	***NESCP***
1	0.80	0.88	0.43	0.51
2	0.97	1.00	0.19	0.19
3	0.92	0.94	2.36	2.69
4	0.96	0.96	1.36	1.65
5	0.94	0.96	0.91	1.09
6	0.90	0.96	0.17	0.17
7	0.94	0.94	1.69	2.20
8	0.86	0.96	1.68	2.17
9	0.92	0.94	0.32	0.38
10	0.92	0.96	1.26	1.65
11	0.94	1.00	0.97	1.10
12	0.88	0.94	0.32	0.42
13	0.88	0.90	0.96	1.10
14	0.96	0.98	0.99	1.25
15	0.82	0.86	0.18	0.25
16	0.84	0.90	0.80	1.12
17	0.88	0.94	0.91	1.12
18	0.84	0.88	0.26	0.307
19	0.94	1.00	0.18	0.19
20	0.90	0.92	1.65	1.90
21	0.92	0.92	0.56	0.65
22	0.86	0.92	0.75	0.87
23	0.94	0.96	1.13	1.65
24	0.90	0.94	0.58	0.75
25	0.90	0.90	1.30	1.52
26	0.94	0.96	0.88	1.03
27	0.96	0.96	0.49	0.64
28	0.96	0.98	0.84	0.90
29	0.88	0.90	1.63	1.85

Table D.4: Validy and efficiency comparison between OSSCP and NESCP per time series category.

over 10 weeks time windows (as described in Section D.5.3). The moving average coverage does not necessarily stabilize at the target 90% but moves up and down the dotted black line. While such behavior is expected (CP does not guarantee conditional coverage), it is intensified by the non-exchangeability of the data, i.e., the presence of distribution shifts in the time series. For example in time series 8, we can notice a heavy drop of the sliding average coverage of the PI with OSSCP to barely reach 63% at a certain time. The distribution shift is better handled by NESCP thanks to the weighted correction of nonconformity scores; the PI's moving average

coverage stays systematically over 82%. The counterpart is the adaptive inflation of the PI width to cope with the data drift.

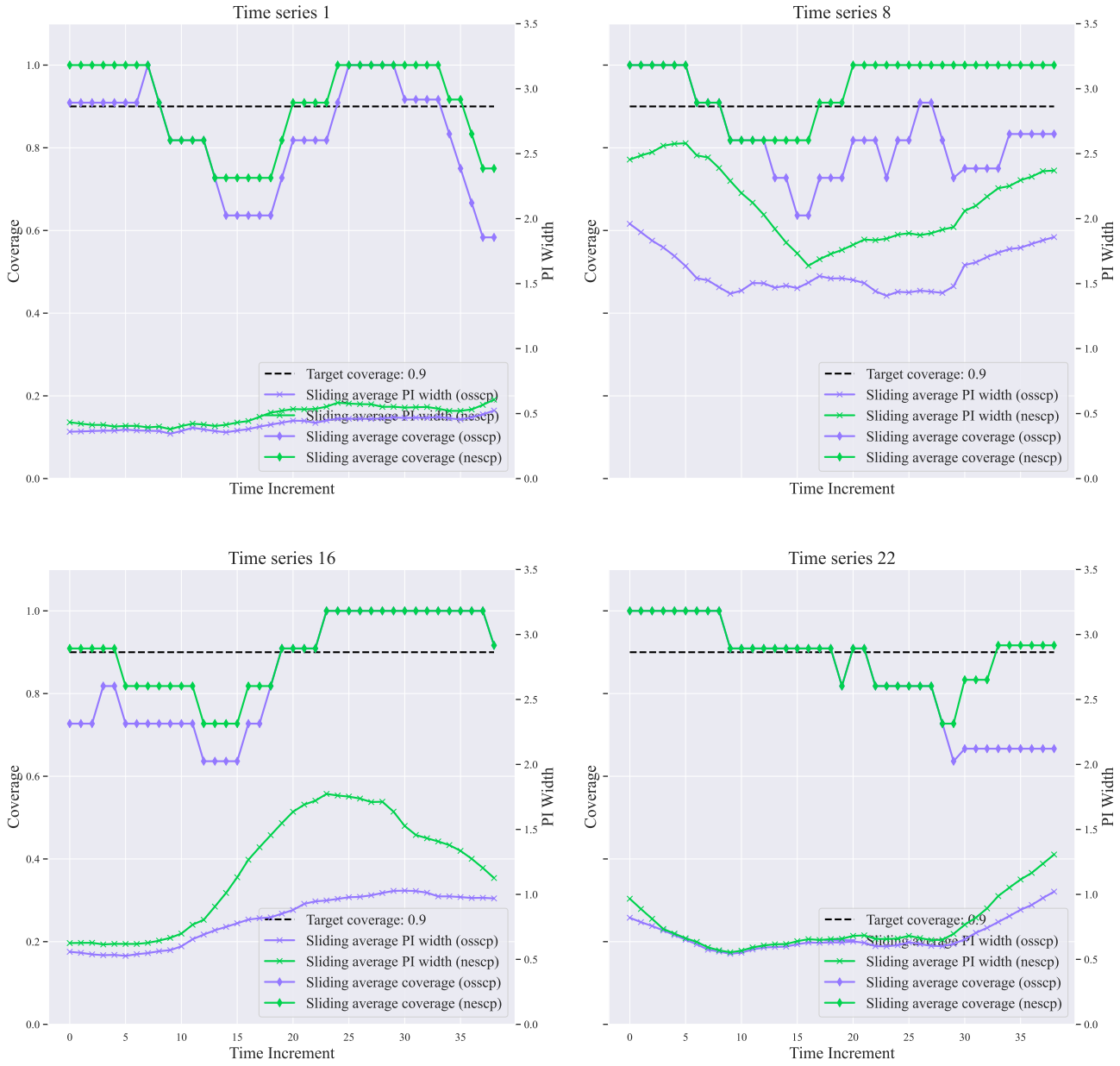


Figure D.4: Moving average coverage and width of PI for some categories with OSSCP and NESCP.

In many situations, our weighted calibration results in a conservative coverage. In Table D.4, we can observe that the marginal coverage of NESCP is usually higher than the target 90%

and sometimes close to 100%. This is directly linked to the exponential decay weight function used on the nonconformity scores during calibration. While it seems as a natural choice to give more "importance" to recent data points when dealing with time series, the way we quantify this "importance" is relevant for the efficiency of the PI. Domain knowledge could be pertinent to anticipate the future trends of distribution shifts and therefore avoid larger PIs that can be somewhat uninformative.

D.7. Conclusion

The subject of CP applied to time series was introduced in 2021. Our previous results are found in the report of the project EC3, Action Sheet 5, released in 2022, and the paper by Mendil et al. [2022]. There, we used a cumulative time window to obtain five large datasets: for each dataset, we fit only one base predictor, conformalize it on a large set of calibration data (covering multiple months or years) and finally we ran the inferences on a large batch of test data. Unsurprisingly, the introduction of the OSSCP scheme greatly improves the performances of the methods tested in 2021. The same algorithms (CQR, etc.) are now refitted and reconformalized after each inference, which happens at every time step (each week). This simple modification, allows the model to be always up-to-date with the latest data. Of course, this scheme is less straightforward to implement for high-frequency data: if we had to run an inference every second, for instance, then we could need to run multiple inferences (i.e. many seconds or minutes) before being able to update and reconformalize our predictors.

With our experiments we also showed how the methods built for time series and to handle distributions shifts (EnbPI, NESCP etc.) can further improve the empirical validity of CP.

Chapter E

Data Subsampling for Bayesian Neural Networks

Contents

E.1. Introduction

Supervised learning is the machine learning task of learning a complex mapping between input and output data. Learning this input-output relation enables the possibility to predict a future output data (yet unknown) based only on the input data. In practice, there is almost always a gap between a model's predictions and the actual observed data: we want to characterize this gap which we call *uncertainty*. The following section shows how to estimate the prediction uncertainty in order to design robust and trustworthy machine learning models. This uncertainty can be decomposed into epistemic and aleatoric uncertainties.

Epistemic uncertainty comes from a lack of knowledge of the real/physical model that generated the data. This error can therefore be reduced by accumulating more data and by increasing the flexibility of the model (the number of parameters for example). These two approaches roughly correspond to a reduction of the variance and the bias of the AI model. However, the reduction of this uncertainty cannot replace its measurement and estimation. Bayesian methods and ensemble methods are currently the most popular approaches for quantifying epistemic uncertainty.

We suggest to use a Monte Carlo estimation of this uncertainty. It consists in sampling the parameters of the learning model, a method that remains the gold standard in uncertainty estimation. We have developed in the following a new alternative for estimating uncertainty using Bayesian Neural Networks.

Moreover, the predictive uncertainty is not exclusively determined by the bias and variance introduced by the representation of the AI model. Indeed, if the phenomenon under study includes a random component, it is impossible to establish an exact deterministic correspondence between input and output data. In this case, classical supervised models are doomed to provide an average prediction whose deviation from the observed value will be non-zero.
This aleatoric uncertainty can sometimes be neglected (signal and noise separation approach). In the case where the stochastic component completely dominates the physical phenomenon, it is no longer possible to neglect it.

For instance, within the framework of meteorological predictions, models often offer a single temperature prediction to the user (random component neglected) whereas they offer a percentage of precipitation (random component taken into account).

In the next section, we will cover the dedicated tools developed in order to estimate the prediction uncertainty within the scope of supervised learning. This estimation is performed by using generative models that allow to approximate the data distribution. With this tool, it is therefore possible to associate prediction and uncertainty in a regression problem.

Markov Chain Monte Carlo (MCMC) algorithms do not scale well for large datasets leading to difficulties in Neural Network posterior sampling. In this document, we apply a generalization of the Metropolis Hastings algorithm that allows us to restrict the evaluation of the likelihood to small mini-batches in a Bayesian inference context. Since it requires the computation of a so-called "noise penalty" determined by the variance of the training loss function over the mini-batches, we refer to this data subsampling strategy as Penalty Bayesian Neural Networks – **PBNNs**. Its implementation on top of MCMC is straightforward, as the variance of the loss function merely reduces the acceptance probability. Comparing to other samplers, we empirically show that PBNN achieves good predictive performance for a given mini-batch size. Varying the size of the mini-batches enables a natural calibration of the predictive distribution and provides an inbuilt protection against overfitting. We expect PBNN to be particularly suited for cases when data sets are distributed across multiple decentralized devices as typical in federated learning.

Its implementation and validation have been tested on a time series prediction problem (Air Liquide use case, see section E.6). In conclusion, during this work we have evaluated the prediction uncertainty by design (both aleatoric and epistemic) on an industrial use case.

Robust & Embeddable - Deep Learning by Design As a reminder, this document follows a batch 1 deliverable from Project EC4 "Trustworthiness by Design". This previous report contains a broad introduction to the Uncertainty Quantification by Design. In particular, it introduces the aleatoric and epistemic uncertainties, the connections between supervised and unsupervised tasks, the bias and variance epistemic uncertainty decomposition, a state of the art over predictive uncertainty quantification in particular for supervised learning tasks containing multiple reviews, a discussion about out of distribution samples and use case applications.

E.2. Preliminaries

In the following we consider a vector θ that describes the parameters (weights and biases) of a Neural Network. We define $p(\theta)$ as a prior distribution over this set of parameters. Commonly used priors are Gaussian prior and Laplace prior that correspond respectively to an L2 and a L1 regularization of the vector θ. We refer as $p(y|x,\theta)$ the probability of a data item y given a data item x and parameter θ. As an example, we aim at sampling the posterior of a Neural Network designed for a supervised task. The posterior distribution over the parameters given a set of data can be written as $p(\theta|\mathscr{D}) \propto p(\theta)\prod_{i=1}^{N} p(y_i|x_i,\theta)$ where $\mathscr{D} = \{(y_i,x_i)\}_{i=1}^{N}$. Up to a constant, the log of the posterior can be written as a loss

$$\mathscr{L}_{\mathscr{D}}(\theta) = -\log p(\theta) - \sum_{i=1}^{N} \log p(y_i|x_i,\theta) \tag{E.1}$$

where the last term corresponds to the Negative Log-Likelihood (NLL). This is an illustrative choice that does not reduce the generality of PBNN as we could have also considered an unsupervised setup where $\mathscr{D} = \{(x_i)\}_{i=1}^{N}$ and $\mathscr{L}_{\mathscr{D}}(\theta) = -\log p(\theta) - \sum_{i=1}^{N} \log p(x_i|\theta)$.

Note that in the following $\mathscr{D}$ indicates precisely the ensemble of datum (y_i,x_i) used for the loss computation $\mathscr{L}_{\mathscr{D}}(\theta)$ in the equation E.1. In particular, this data set can be a sub sample (mini-batch) of the larger data set containing all known data points of the training set.
As a reminder, the usual maximum likelihood estimate corresponds to

$$\begin{aligned} \theta^{\mathrm{MLE}} &= \arg_\theta \max \left[\log p(\mathscr{D}|\theta)\right] \\ &= \arg_\theta \max \left[\sum_i \log(p(y_i|x_i,\theta))\right] \end{aligned} \tag{E.2}$$

where $p(\mathscr{D}|\theta)$ is differentiable for a neural network and the maximum is evaluated through gradient descent techniques.

Example: Mean Squared Error cost function and L2 regularization As an illustration, we consider a Gaussian prior that writes

$$p(\theta) \propto \exp(-\mu\|\theta\|_2^2) = \exp(-\mu\sum_{i=1}^{K}\theta_i^2) \tag{E.3}$$

with θ_i the K parameters of the model. Using this expression we obtain

$$\begin{aligned}\mathscr{L}_{\mathscr{D}}(\theta) &= -\log p(\theta) - \sum_{i=1}^{N} \log p(y_i|x_i,\theta) \\ &= -\sum_{i=1}^{N} \log\left[p(y_i|x_i,\theta)\right] + \mu \sum_{k=1}^{K} \theta_i^2 + \text{cst}_1\end{aligned} \tag{E.4}$$

Suppose that we model the data by a Gaussian distribution with a fixed variance (homoscedastic):

$$p(y_i|x_i,\theta) \propto \exp(-(y_i - f_\theta(x_i))^2) \tag{E.5}$$

with $f_\theta(x_i)$ being an arbitrary function (linear, polynomial, Neural Network etc...).

We then obtain

$$\mathscr{L}_{\mathscr{D}}(\theta) = \sum_{i=1}^{N} \left[(y_i - f_\theta(x_i))^2\right] + \mu \sum_{k=1}^{K} \theta_i^2 + \text{cst}_2 \tag{E.6}$$

which corresponds to the standard MSE loss in a regression task with a L2 regularization term.

E.3. Related work

We introduce in this section some relevant literature that studied how to take into account a noisy gradient estimate of $\nabla_\theta \mathscr{L}_{\mathscr{D}}(\theta)$ computed from a subset of the data. The link between noisy gradient and BNN posterior sampling is detailed in section E.4.2.

Stochastic Gradient Langevin Dynamics Max Welling and Yee Whye Teh (2011) Welling and Teh showed that the iterates θ_t will converge to samples from the true posterior distribution as they anneal the stepsize by adding the right amount of noise to a standard stochastic gradient optimization algorithm. This is known as the Stochastic Gradient Langevin Dynamics (SGLD) where the parameter update is given by

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta_t \nabla_\theta \widetilde{\mathscr{L}_{\mathscr{D}}}(\theta_t) + \sqrt{2\eta_t}\varepsilon_t \\ \widetilde{\mathscr{L}_{\mathscr{D}}}(\theta_t) &= -\log p(\theta) - \frac{N}{n}\sum_{i=1}^{n} \log p(y_i|x_i,\theta)\end{aligned} \tag{E.7}$$

where $\eta_t \in \mathbb{R}^+$ is a learning rate and ε_t is a centered normally distributed random vector. No rejection step is required for a vanishing step size. The positive whole number n corresponds to the size of the subsampled mini-batch. Chen (2014) Chen et al. [2014] later extended this idea to HMC sampler.

Noisy Posterior Sampling Bias Due to a potentially high variance of the stochastic gradients $\nabla_\theta \widetilde{\mathscr{L}_{\mathscr{D}}}(\theta)$, Brosse (2018) Brosse et al. [2018] showed that the SGLD algorithm has an invariant probability measure which in general significantly departs from the target posterior for any non vanishing stepsize η. Furthermore, a recent work from Garriga-Alonso (2020) Garriga-Alonso and Fortuin [2021] suggests that recent versions of SGLD implementing an additional Metropolis-Hastings rejection step do not improve this issue, because the resulting acceptance probability is likely to vanish too.

Failures of Data Set Splitting Inference Other works have exploited parallel computing to scale Bayesian inference to large datasets by using a two-step approach. First, a MCMC computation is run in parallel on K (sub)posteriors defined on data partitions following $p(\theta|\mathscr{D}) \propto \prod_{i=1}^{K} p(\theta)^{1/K} p(\mathscr{D}_i|\theta)$. Then, a server combines local results. While efficient, this framework is very sensitive to the quality of subposterior sampling as showned by de Souza (2022) Souza et al. [2022].

E.4. Penalty Bayesian Neural Network

E.4.1 Unbiased Posterior Sampling

We suppose that the data set points (y_i, x_i) are sampled from an unknown distribution $p(y,x)$ that can provide an infinite number of independent and identically distributed (i.i.d) random data points. This allows us to properly define a true unbiased loss $\mathscr{L}(\theta)$ as the mean over all possible data sets $\mathscr{L}(\theta) \simeq \mathscr{L}_{\mathscr{D}}(\theta)$ with

$$\mathscr{L}(\theta) = -\log p(\theta) - N\mathbb{E}_{(y_i,x_i)\sim p(y,x)}[\log p(y_i|x_i,\theta)] \tag{E.8}$$

given a fixed size N of a random data set $\mathscr{D}$. In order to sample from this log-posterior, we note that detailed balance is a sufficient but not necessary condition to ensure that a Markov process possesses a stationary distribution proportional to $e^{-\mathscr{L}(\theta)}$. Concretely, the detailed balance can be written as

$$A(\theta,\theta')q(\theta|\theta')e^{-\Delta(\theta',\theta)} = A(\theta',\theta)q(\theta'|\theta) \tag{E.9}$$

where $A(\theta',\theta)$ corresponds to the acceptance of the move from θ to θ' and $q(\theta'|\theta)$ is a proposal distribution. The loss difference writes

$$\Delta(\theta',\theta) = \mathscr{L}(\theta') - \mathscr{L}(\theta) \tag{E.10}$$

In the following, we assume that the true loss difference $\Delta(\theta',\theta)$ is unknown, and loss differences can only be estimated based on random data sets $\mathscr{D}$. Then we can introduce a random variable $\delta(\theta',\theta)$ providing an unbiased estimator of $\Delta(\theta',\theta)$ which we assume as normally distributed

$$\delta(\theta',\theta) \sim \mathscr{N}(\Delta(\theta',\theta), \sigma^2(\theta',\theta)) \tag{E.11}$$

The variance $\sigma^2(\theta',\theta)$ typically decreases with the size N of the random data sets $\mathscr{D}$.

This noisy loss $\delta(\theta',\theta)$ introduces a bias in the posterior sampling if not correctly taken into account. In the context of statistical physics and computational chemistry, Ceperley and Dewing (1999) Ceperley and Dewing [1999] have generalized the Metropolis-Hastings random walk algorithm to the situation where the loss is noisy and can only be estimated. They showed that it is possible to still sample the exact distribution even with very strong noise by modifying the acceptance probability and applying a noise penalty $-\sigma^2(\theta',\theta)/2$ to the loss difference in the acceptance ratio A such that

$$A(\delta,\theta,\theta') = \min\left(1, e^{-\delta(\theta',\theta)-\sigma^2(\theta',\theta)/2}\right) \tag{E.12}$$

One can then show that detailed balance is satisfied on average

$$\begin{aligned} &\int d\delta A(\delta,\theta,\theta')q(\theta|\theta')\mathscr{N}(\delta;\Delta(\theta',\theta),\sigma^2(\theta',\theta))e^{-\delta} \\ &= \int d\delta A(\delta,\theta',\theta)q(\theta'|\theta)\mathscr{N}(\delta;\Delta(\theta,\theta'),\sigma^2(\theta,\theta')) \end{aligned} \tag{E.13}$$

which is sufficient condition for the Markov chain to sample the unbiased distribution in the stationary regime. As shown by Ceperley and Dewing in the special symmetric proposal distribution $q(\theta'|\theta)$ case, the acceptance satisfies the equation E.13 with

$$\begin{aligned} &\int d\delta A(\delta,\theta,\theta')\mathscr{N}(\delta;\Delta(\theta',\theta),\sigma^2(\theta',\theta)) \\ &= \frac{e^{-\Delta}}{2}\,\mathrm{erfc}\left(\frac{1}{2\sqrt{\eta}}(\eta-\Delta)\right) + \frac{1}{2}\,\mathrm{erfc}\left(\frac{1}{2\sqrt{\eta}}(\eta+\Delta)\right) \end{aligned} \tag{E.14}$$

where erfc is the complimentary error function. The penalty method can further be extended to a non symmetric proposal distribution $q(\theta'|\theta)$ used in algorithm 1.

Algorithm 1 PBNN Metropolis Adjusted Algorithm

$\theta_t \leftarrow \theta_0$
for $t \leftarrow 0$ to T **do**
  $\theta' \sim q(\theta'|\theta_t)$
  $A(\delta,\theta',\theta_t) \leftarrow \min\left(1, \frac{q(\theta_t|\theta')}{q(\theta'|\theta_t)} e^{-\delta(\theta',\theta_t)-\sigma^2(\theta',\theta_t)/2}\right)$
  $u \sim \mathscr{U}(0,1)$
  if $u \leq A(\delta,\theta',\theta_t)$ **then**
    $\theta_{t+1} \leftarrow \theta'$
  else
    $\theta_{t+1} \leftarrow \theta_t$
  end if
  $t \leftarrow t+1$
end for

From equation E.12 one can immediately recognize the drawback of PBNN leading to an exponential suppression of the acceptance since the variance $\sigma^2(\theta',\theta)$ is always non negative. Note further, that in the case of BNN posterior sampling, $\sigma^2(\theta',\theta)$ is in general not known either, and can only be estimated, too. Whereas it is possible to extend the scheme to account for noisy variances Ceperley and Dewing [1999], we will not pursue this here.

Let us stress that the penalty term serves to exactly account for the uncertainty in calculating the loss $\mathscr{L}(\theta)$ of equation E.8 for a finite number of random data. However, it does not address the actual uncertainty introduced by setting $\mathscr{L}(\theta) \approx \mathscr{L}_{\mathscr{D}}(\theta)$, e.g. equation E.8 and equation E.1.

E.4.2 Langevin Dynamic Penalty

Choosing a non symmetric proposal distribution $q(\theta'|\theta)$ can speed up the mixing of the Markov Chain and help PBNN scale to larger systems by maximizing the acceptance $A(\theta',\theta)$. We first

consider the situation where the proposal distribution depends only on the two states θ and θ' and not on any given mini-batch $\mathscr{D}$. In the absence of noise, the Metropolis-Hastings acceptance writes

$$\begin{aligned} A(\theta',\theta) &= \min\left(1, \frac{q(\theta|\theta')}{q(\theta'|\theta)} e^{-\Delta(\theta',\theta)}\right) \\ &\approx \min\left(1, \frac{q(\theta|\theta')}{q(\theta'|\theta)} e^{-(\theta-\theta')\cdot(\nabla_\theta \mathscr{L}(\theta')+\nabla_\theta \mathscr{L}(\theta))/2}\right) \end{aligned} \tag{E.15}$$

where we have Taylor expanded the loss $\mathscr{L}(\theta)$ around θ' assuming a sufficiently small step from θ to θ'. The maximization of $A(\theta',\theta)$ leads to a Langevin equation where the gradient of the loss introduces a drift in the Gaussian proposal distribution

$$q(\theta'|\theta) = \mathscr{N}(\theta'; \theta - \eta\nabla_\theta \mathscr{L}(\theta), 2\eta) \tag{E.16}$$

Sampling a new state θ' from the proposal distribution $q(\theta'|\theta)$ corresponds exactly to drawing a centered reduced normal random variable ε, and computing

$$\theta' = \theta - \eta\nabla_\theta \mathscr{L}(\theta) + \sqrt{2\eta}\varepsilon \tag{E.17}$$

The non-trivial term in the Metropolis-Hastings acceptance then writes

$$\begin{aligned} &\log\left(\frac{q(\theta|\theta')}{q(\theta'|\theta)} e^{-\Delta(\theta',\theta)}\right) \\ &\quad = \frac{-1}{4\eta}\|\eta\left(\nabla_\theta \mathscr{L}(\theta') + \nabla_\theta \mathscr{L}(\theta)\right) - \sqrt{2\eta}\varepsilon\|^2 \\ &\quad + \frac{1}{4\eta}\|\sqrt{2\eta}\varepsilon\|^2 - \mathscr{L}(\theta') + \mathscr{L}(\theta) \end{aligned} \tag{E.18}$$

This algorithm is called Metropolis-Adjusted Langevin Algorithm (MALA). We write $\eta = \sigma^2/2$ as an analogy with the learning rate in a gradient descent.

Algorithm 2 Metropolis-Adjusted Langevin Algorithm (MALA)

$t \leftarrow 0$
$\theta_t \leftarrow \theta_0$
for N **do**
  $\varepsilon_t \sim \mathscr{N}(\mathbf{0}, \mathbb{I}_d)$
  $\theta' \leftarrow \theta_t - \eta\nabla_\theta \mathscr{L}_{\mathscr{D}}(\theta_t) + \sqrt{2\eta}\varepsilon_t$
  $A(\theta',\theta_t) \leftarrow \min\left(1, \frac{p(\theta')q(\theta|\theta')}{p(\theta)q(\theta'|\theta)}\right)$
  $u \leftarrow$ sample $\mathscr{U}(0,1)$
  if $u \leq A(\theta',\theta_t)$ **then**
    $\theta_{t+1} \leftarrow \theta'$
  else
    $\theta_{t+1} \leftarrow \theta_t$
  end if
  $t \leftarrow t+1$
end for

The Unadjusted Langevin Algorithm (ULA) corresponds to the same algorithm without the MH acceptance step in the vanishing $\eta \to 0$ limit.

Algorithm 3 Unadjusted Langevin Algorithm (ULA)

$t \leftarrow 0$
$\theta_t \leftarrow \theta_0$
for N **do**
  $\eta_t \leftarrow$ a vanishing value as $t \to \infty$
  $\varepsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_d)$
  $\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla_\theta \mathscr{L}_{\mathscr{D}}(\theta_t) + \sqrt{2\eta_t}\varepsilon_t$
  $t \leftarrow t+1$
end for

Note: we notice that for $\eta_t \to 0$ any noise coming from the energy in $\eta_t \nabla_\theta \mathscr{L}_{\mathscr{D}}(\theta_t)$ is dominated by the random walk noise $\sqrt{2\eta_t}\varepsilon_t$.

In order to use a noisy gradient $\nabla_\theta \mathscr{L}(\theta) \simeq \nabla_\theta \widetilde{\mathscr{L}_{\mathscr{D}}}(\theta)$ as an approximation of the Gaussian mean's drift, SGLD Welling and Teh requires a vanishing learning rate to dominate the noise and maximize the acceptance. On the other hand, one could design an optimized proposal distribution $q(\theta|\theta')$ and set a non zero step size while computing the full Metropolis-Hastings acceptance

$$A(\theta', \theta) = \min\left(1, \frac{q(\theta_t|\theta')}{q(\theta'|\theta_t)} e^{-\delta(\theta', \theta_t) - \sigma^2(\theta', \theta_t)/2}\right) \tag{E.19}$$

The PBNN's noise penalty explicitly targets the bias introduced by a noisy loss. In fact, in the equation E.19, the corresponding Monte Carlo loss minimization (i.e. introducing a zero temperature limit) corresponds to finding the set of parameters θ that minimizes both the noisy regularized loss and its associated uncertainty.

For large size models, biased samplers like the Unrestricted Langevin Algorithm (ULA) are known to be very effective as they skip the rejection step i.e set $A(\theta', \theta) = 1$ for a sufficiently small step size η resulting in an unrestricted Langevin sampling as

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathscr{L}(\theta_t) + \sqrt{2\eta}\varepsilon_t \tag{E.20}$$

In order to model a noisy estimate of the loss, it is tempting to replace the drift $\eta \nabla_\theta \mathscr{L}(\theta_t)$ with an unbiased estimator such that

$$\eta \nabla_\theta \mathscr{L}(\theta) = \eta \nabla_\theta \widetilde{\mathscr{L}_{\mathscr{D}}}(\theta) + \eta \sigma(\theta) \tag{E.21}$$

where $\widetilde{\mathscr{L}_{\mathscr{D}}}(\theta)$ is defined in the equation E.7. For a vanishing step size $\eta \to 0$, one may then expect that the additional noise term $\eta\sigma(\theta)$ gets negligible compared to the random noise of order $\eta^{1/2}$ in the equation E.20. However, the uncertainty of the loss gradient $\sigma(\theta)$ does in general not result in white noise, but is correlated between different parameters θ. For non-vanishing η the noisy loss gradient can thus trigger a significant departure from the target posterior, see also Brosse et al. [2018].

As we will see in the numerical experiments section, PBNN's ability to evaluate the likelihood over small mini-batches even in the presence of a strong noise allows us to calibrate the Bayesian predictive distribution. This is especially convenient in comparison with usual BNNs as the regularization is handled solely by the prior distribution $p(\theta)$. Commonly used uninformative priors can lead to poor performances as they do not target explicitly overfitting but rather the complexity of the model i.e. L2 and L1 penalties. As a reminder, other conventional methods such as early stopping are not compatible with the Bayesian approach developed for BNN.

E.4.3 Noise Penalty Estimation

As showed in equation E.12 in the case of a symmetric proposal distribution $q(\theta'|\theta)$, the energy difference $\delta(\theta',\theta_t)$ has to dominate the always positive variance $\sigma^2(\theta',\theta_t)/2$ in order to obtain a reasonable acceptance $A(\delta,\theta',\theta_t)$. However, in practice a noise penalty usually strongly dominates any gain in energy from θ_t to θ' if the energy difference is computed only on a single small mini-batch $\mathscr{D}$. This leads to an exponentially suppressed acceptance and long correlation times of the MCMC.

To prevent this situation, we define $\delta(\theta',\theta)$ as an empirical average over the loss difference

$$\delta(\theta',\theta) = \frac{1}{M}\sum_{j=1}^{M}\left(\mathscr{L}_{\mathscr{D}_j}(\theta') - \mathscr{L}_{\mathscr{D}_j}(\theta)\right) \tag{E.22}$$

where $\mathscr{D}_j$ corresponds to randomly chosen mini-batches. By definition the average is an unbiased estimator such that $\mathbb{E}[\delta(\theta',\theta)] = \Delta(\theta',\theta)$. We notice that the central limit theorem ensures that $\delta(\theta',\theta)$ is normally distributed in the limit of large M as required by equation E.11.

The variance of the random variable $\delta(\theta',\theta)$ strictly decreases with the number of mini-batches M since $\sigma^2(\theta',\theta) = \sigma^2_{\mathscr{D}}(\theta',\theta)/M$ where $\sigma^2_{\mathscr{D}}$ corresponds to the expected variance of a single loss difference computed over a mini-batch $\mathscr{D}$. Both $\sigma^2_{\mathscr{D}}$ and σ^2 are unknown, but we can compute an estimate of $\sigma^2(\theta',\theta) \simeq \chi^2(\theta',\theta)$ using an unbiased chi-squared estimator

$$\begin{aligned}&\chi^2(\theta',\theta)\\ &= \frac{1}{M(M-1)}\sum_{j=1}^{M}\left(\mathscr{L}_{\mathscr{D}_j}(\theta') - \mathscr{L}_{\mathscr{D}_j}(\theta) - \delta(\theta',\theta)\right)^2\end{aligned} \tag{E.23}$$

In the following, we do not take into account the error over the estimation of the variance $\sigma^2(\theta',\theta)$ which corresponds to the hypothesis that variations of χ^2 as a function of θ' and θ largely dominate over the noise. Leading order corrections in this noise are discussed in Ceperley and Dewing [1999].

A second approximation can be introduced in case of a limited access to the data: drawing M mini-batches with replacement artificially decreases the variance at the cost of introducing a bias (i.e. violates the i.i.d. hypothesis of the energy differences).

E.5. Experiments

E.5.1 Data Set

We study the performance of PBNN on a synthetic data set that contains the positions of a double pendulum over a simulated time t. These positions are obtained by integrating Euler-Lagrange equations casted as ordinary differential equations. We turn a time series forecasting problem into a supervised regression task. We model the distribution $p(y|x)$ where $y \in \mathbb{R}^4$ corresponds to the four Cartesian coordinates of the two masses of the double pendulum and $x \in \mathbb{R}^{4*5}$ are 5 given y past positions of the masses. The data set $\mathscr{D} = \{(y_i, x_i)\}_{i=1}^{N}$ inputs x_i write

$$x_t \equiv (y_{t-20}, \ldots, y_{t-24}) \tag{E.24}$$

where t is the discrete simulation time. The system is strongly chaotic such that learning on a given limited data set can lead to strong predictive overconfidence as suggested in figure E.1. We thus expect the noise penalty to play an important role on the realization of a supervised task based on this data set.

The code that generates the double pendulum dataset is available following this link: https://gitlab.com/eijikawasaki/double-pendulum

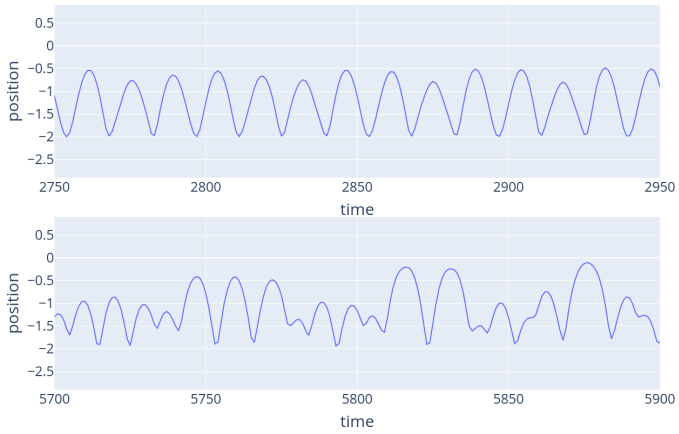


Figure E.1: Pendulum data set extracts from a single simulation run. The blue curve corresponds to one of the Cartesian coordinates of one of the masses in function of time. As an example, the behavior on the bottom (part of the test data set) is hard to predict knowing only the data from the top (part of the training data).

E.5.2 Benchmark setup

The goal of this section is to compare the performances between PBNN and other BNN that do not include any noise penalty. We show empirically that the PBNN obtains good predictive performances even while evaluating the loss on small mini-batches and therefore introducing a strong noise. On the other hand, as expected PBNN has a much lower MCMC acceptance rate.

We also compare the PBNN to SGLD which is designed to take into account a stochastic noise in the loss computation.

In the following, we sample the posterior based on MCMC random walkers where $q(\theta'|\theta_t)$ is a symmetric proposal density. We use a Gaussian distribution centered around θ_t and adjust its variance to ensure the ergodicity of the Markov process. We model the data distribution with a single multivariate Gaussian likelihood $p(y|x,\theta) = \mathcal{N}(y;\mu_\theta(x),\Sigma^2_\theta(x))$ where $\mu_\theta(x) \in \mathbb{R}^4$ and $\Sigma^2_\theta(x)$ is a positive diagonal covariance matrix parameterized by $\sigma^2_{\theta,d}(x)$ where $d \in \{1,2,3,4\}$. This model is thus heteroscedastic: we use a Mixture Density Network Bishop [1994] such that both $\mu_\theta(x)$ and $\Sigma^2_\theta(x)$ are outputs of a neural network that takes x as an input. We observe empirically that a homoscedastic model based on a Mean Squared Error loss has a noise penalty that is several orders of magnitude smaller than its heteroscedastic counterpart.

The loss of a NN is known to have numerous local minima and we don't use in the experiment any transition kernel for PBNN designed to jump between separate minima. We thus limit our study on the impact of the penalty method on a relatively small model that is easier to sample. The dimension of the vector parameter θ is equal to 419 as we consider a NN with 2 hidden layers each containing 10 neurons. The data consists in 9975 data points sequentially (i.e. not randomly) split into a 2992 points as a training data and 6983 points as a test data set. We use a Gaussian uninformative prior $p(\theta) \propto e^{-\lambda\|\theta\|_2^2}$ corresponding to a tiny L2 regularization of the NN parameters of magnitude $\lambda = 10^{-5}$. During the posterior sampling, only the training data is used in order to compute the loss $\mathscr{L}_{\mathscr{D}}(\theta)$ and thus both $\delta(\theta',\theta)$ and $\chi^2(\theta',\theta)$.

The performance of the prediction (based on the inferred parameters θ sampled from the posterior) is measure by the the average Negative-Log-Likelihood (NLL) that we define as

$$\overline{\text{NLL}_{\mathscr{D}}} = -\frac{1}{L}\sum_{i=1}^{L} \log\left(\frac{1}{J}\sum_{j=1}^{J} p(y_i|x_i,\theta^{(j)})\right) \tag{E.25}$$

where the data set $\mathscr{D}$ of size L corresponds either to the train or the test sets. $\theta^{(j)}$ are J i.i.d. samples of the MDN parameters obtained from the Markov chain. Note that this measure should not depend on θ since the MDN prediction is marginalized over the posterior parameters distribution.

The mini-batch size N in equation E.1 determines the target posterior distribution that is sampled and therefore changes the value of $\overline{\text{NLL}_{\mathscr{D}}}$. For a constant uninformative prior, decreasing N corresponds to increasing the variance of the predictive function. This is a straightforward consequence of Bayes theorem using an uninformative Gaussian prior with a huge variance. In order to compare the performances of the prediction of a BNN and a reference standard "vanilla" BNN that does not use mini-batches, we compute one-sigma confidence intervals as drawn in the figure E.2. As an example, we compare them to a Gaussian predictive distribution and target an expected coverage of approximately 68.2%. To check the accuracy of the UQ method, we compute the Average Coverage Error (ACE) defined in the equation E.26.

$$\text{ACE} = \left|68.2\% - \frac{1}{4L}\sum_{i=1}^{L}\sum_{d=1}^{4}\rho_{i,d}\right| \qquad \text{with } \rho_{i,d} = \begin{cases} 1 & \text{if } y_{i,d} \in \left[\mu^*_d(x_i) - \sigma^*_d(x_i), \mu^*_d(x_i) + \sigma^*_d(x_i)\right] \\ 0 & \text{otherwise} \end{cases} \tag{E.26}$$

where μ_d^* and σ_d^* are the mean and the standard deviation in one of the four dimension d of the empirical predictive distribution as defined in equation E.27.

E.5.3 Numerical Results

Table E.1: Performance benchmark. The top and bottom groups of models correspond to the predictive performance based on a likelihood weight corresponding respectively to $N = 2992$ and $N = 60$.

Model	Test $\overline{\text{NLL}_{\mathscr{D}}}$	Train $\overline{\text{NLL}_{\mathscr{D}}}$	Test ACE
Vanilla BNN	-4.11 ± 0.01	-5.36 ± 0.01	$7.1\% \pm 0.3\%$
Tempered BNN	-1.96 ± 0.08	-2.05 ± 0.10	$3.9\% \pm 1.1\%$
Batched BNN	-1.68 ± 0.17	-1.74 ± 0.19	$1.7\% \pm 1.6\%$
pseudo-SGLD	-2.35 ± 0.10	-2.48 ± 0.11	$4.8\% \pm 0.8\%$
PBNN	**-3.91**± 0.07	**-4.83**± 0.09	**0.4**$\% \pm 2.3\%$

From table E.1 we empirically show that PBNN achieves a better overall performance than other biased sub sampled models for a small mini-batch size. As an illustration, figure E.2 shows the error bars of the prediction over a random period of time for each model. The empirical predictive distribution is defined as

$$\mathbb{E}[p(y_t|x_t,\theta)] \simeq \frac{1}{J}\sum_{j=1}^{J} p(y_t|x_t,\theta^{(j)}) \tag{E.27}$$

where t is the simulated time and $\theta^{(j)}$ are J samples obtained by the MCMC computation. The expected value in the equation E.27 is computed over the targeted posteriors which are different for every model studied in the benchmark.

It is important to note here that even for a same likelihood weight N in the Bayes theorem, we do not expect the same prediction between models that use the whole train set, like "vanilla" BNNs and the PBNN as they do not target the same posterior sampling. The vanilla BNN samples a posterior proportional to $e^{-\mathscr{L}_{\mathscr{D}}(\theta)}$ whereas PBNN aims at sampling a posterior proportional to $e^{-\mathscr{L}(\theta)}$ where $\mathscr{L}(\theta)$ is the loss expected for a given size N of mini-batch $\mathscr{D}$ as defined in the equation E.8.

Vanilla BNN We call this first model "vanilla" as it corresponds to a standard MCMC random walk based BNN with no mini batches. Vanilla BNN's acceptance writes $A(\theta',\theta_t) = \min(1, e^{-\mathscr{L}_{\mathscr{D}}(\theta')+\mathscr{L}_{\mathscr{D}}(\theta_t)}))$ where $\mathscr{D}$ contains all the 2992 available train data points. This model thus uses the whole train set to compute the loss difference for each new proposed state θ'. Note that Langevin algorithms such as MALA or HMC all sample the same posterior. The only difference with this model is in their efficiency as they are designed to maximize the MCMC acceptance and the ergodicity of the Markov Chain.

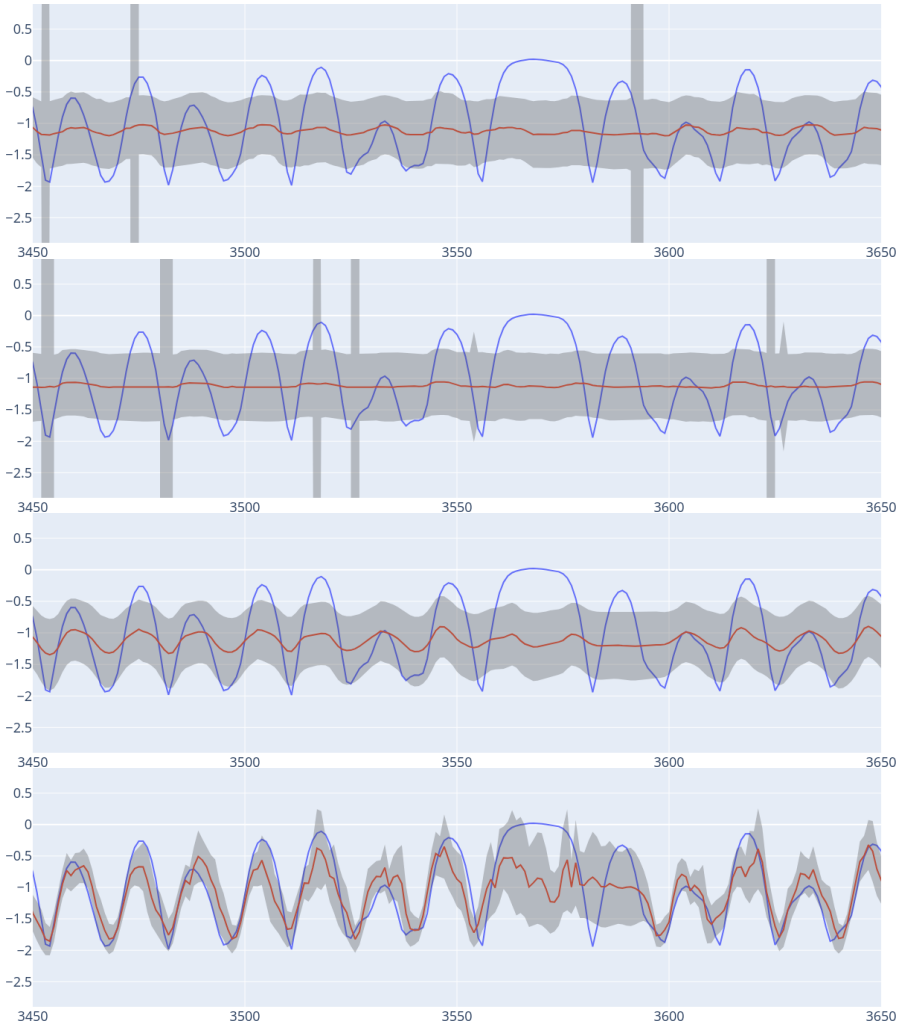


Figure E.2: Models predictions over a test data set example extract. The blue line corresponds to one of the Cartesian coordinates of one of the two masses. Mean models predictions are in red and one standard deviation regions are plotted in grey. The horizontal axis corresponds to the the simulated time t. The four figures from top to bottom correspond respectively to the models: Tempered BNN, Batched BNN, pseudo-SGLD and **PBNN**.

The ACE is approximately zero on the training set and significantly different from zero on the test data set. In the limit of a single mini-batch containing all the training data set, the PBNN coincides with this model. In the following we aim at calibrating the PBNN predictive prediction while maintaining good predictive performances.

Tempered BNN In order to provide a comparison between the usual "vanilla" BNN and a PBNN, we adjust the likelihood weight following

$$-\log p(\theta) - \frac{N}{2992}\sum_{i=1}^{2992}\log p(y_i|x_i,\theta) \tag{E.28}$$

Indeed, we balance the loss to be equal to PBNN's likelihood that uses mini-batches with $N = 60$. This adjusted weight over the likelihood corresponds to the Safe Bayes approach where we vary the weight of the likelihood thanks to a temperature T following $p(\mathscr{D}|\theta)^{1/T}$ as discussed by Wilson (2020) Wilson and Izmailov [2020]. In our case $T > 1$ is known to help under model misspecification as it is the case in the double pendulum Gaussian prediction example.

The resulting temperature $T = 2992/60$ is however such a high temperature that the prior regularization dominates the likelihood for this model. The resulting predictive performance is unsatisfactory as shown in the table E.1.

Batched BNN The acceptance for this model is defined as $A(\delta,\theta',\theta_t) = \min(1, e^{-\delta(\theta',\theta_t)})$ with $\delta(\theta',\theta_t)$ computed with $M = 100$ and $N = 60$. The number of mini-batches used by this model is the same as the number used by the PBNN for a fair comparison. The only difference with the PBNN model is the noise penalty $e^{-\chi^2(\theta',\theta_t)/2}$ in the acceptance. Table E.1 therefore demonstrates the impact of the penalty method, strongly improving the overall performance of the model.

pseudo-SGLD The SGLD algorithm is designed to naturally take into account noise from sub-sampled data. Standard SGLD with a weight $N = 2992$, i.e. the number of training samples, in equation E.7 leads to results that are similar to the Vanilla BNN both in terms of negative loglikelihood and coverage. In this benchmark we want to test the ability of the algorithm to handle a noisy loss. We therefore set $N = n = 60$ with a constant learning rate $\eta = 10^{-5}$ and call the resulting model a pseudo-SGLD. Comparing to PBNN, we observe in both table E.1 and figure E.2 that the noise is too high for the SGLD in this setup. The performance of SGLD could probably be improved by decaying the step size η polynomially as suggested in the literature Welling and Teh. As a reminder, this model has the great advantage of not requiring a rejection step.

PBNN The noise penalty is estimated following equations E.22 and E.23 with $M = 100$ and $N = 60$. The random walk acceptance for PBNN writes

$$A(\delta,\theta',\theta_t) = \min\left(1, e^{-\delta(\theta',\theta_t)-\chi^2(\theta',\theta_t)/2}\right)$$

We have adjusted by hand the mini-batch size $N = 60$ in order to calibrate the models to optimize the test data set coverage.

There is a noticeable gap of PBNN's performance between the train and the test data sets that is not intuitively described in the theory of PBNN. This overfitting is probably caused partially by the access to a limited amount of data. Indeed, during the Monte Carlo sampling, all mini-batches $\mathscr{D}$ are part of the same training data and not i.i.d. sampled from a probability distribution

$p(\mathscr{D})$. In an ideal situation where we could evaluate both δ and χ^2 from i.i.d. samples $\mathscr{D} \sim p(\mathscr{D})$ as required by the equations E.22 and E.23, we expect a lower overfitting effect.

E.5.4 Mini-Batch Size N

For the purpose of the benchmark we have calibrated PBNN error bars in the table E.1 by tuning the mini-batch size N. One can wonder what is the optimal value of N in a general purpose outside the scope of calibration and for a given data set size. Figure E.3 shows that, as expected, the prediction performance measured as the negative loglikelihood $\overline{\text{NLL}_{\mathscr{D}}}$ over the test data set increases with N. On the other hand, the acceptance rapidly drops because the number of mini-batches M in equation E.23 decreases for a constant data set size. In the figure E.3 we indeed notice a linear decrease of the log-acceptance due to the decrease of the number of available mini-batches M. The optimal value of N is therefore a trade-off between reasonable acceptance and good predictive performance.

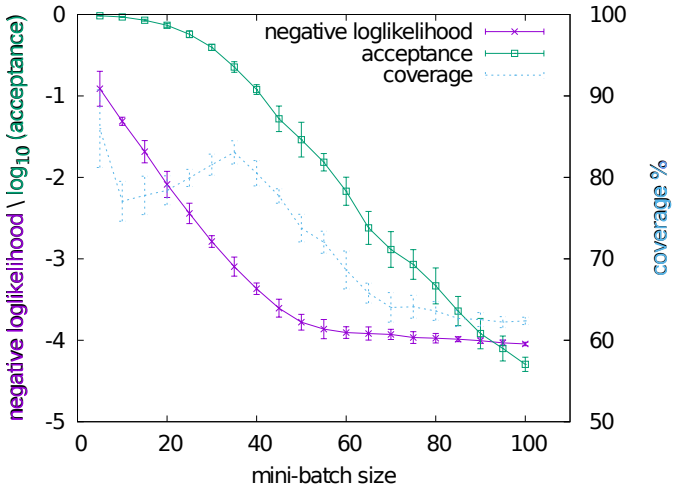


Figure E.3: PBNN performance measured by $\overline{\text{NLL}_{\mathscr{D}}}$ over the test data set, acceptance in base $\log_{10}$ and one sigma coverage in function of the mini-batch size N for a constant prior. The standard deviation continuously decreases in function of the batch size. We notice however that the coverage is not monotonous: it is determined by both the error bars size and by the loglikelihood.

It is important to note in the figure E.3 that different batch sizes result in different coverages for a constant uninformative prior. As we have shown the PBNN predictive distribution can be calibrated. In practical setups as discussed by Hermans (2021) Hermans et al. [2021], it is recommended to compare the expected coverage probability of the predictive distribution defined in equation E.27 to the empirical coverage probability as shown in the equation E.26.

E.6. Use Case: Air Liquide demand forecast

We have developed a python library delivered to the confiance.ai program, it contains:

- a MCMC module that is able to sample a distribution in high dimension that is defined up to a constant
- a Mixture Density Network module to take into account the aleatoric component of the uncertainty
- an Uncertainty Quantification module that samples a Bayesian Neural Network posterior using MCMC

We apply PBNN to the Air Liquide demand forecast use case. Examples are shown in figure E.4. The data batch size as defined in equation E.1 is set to $N = 20$. We compute the error bars coverage as defined in equation E.26.

We compare these results with a naive batched BNN using $N = 20$ with no penalty term. We observe in the figure that the noise penalty has a dramatic effect on the predictive performance. Performance is measured as defined in equation E.25.

The figure E.4 demonstrates that, using PBNN, it is possible to train a Bayesian Neural Network on batched data from the Air Liquide use case. As expected we observe on the left column that standard BNNs obtain bad predictive performances while trained on data sub-samples. In contrast, the right column shows that we obtain good predictive performances with PBNN that use batches of $N = 20$ sub-sampled data points.

The sub-sampling is useful for computational efficiency and also for the uncertainty quantification calibration. Indeed, we have shown that varying N i.e. the size of the batches corresponds to varying the weight of the likelihood in the Bayes equation. Using an uninformative prior, decreasing the mini-batch size corresponds to increasing the size of the error-bars in the figure E.4. It is thus possible to calibrate them to match the expected and empirically observed coverage.

The Air Liquide use case data size is not compatible with large model training. As an outlook, we thus aim in the next project phases to apply the PBNN concepts to data and use case that require bigger models and study the impact of mini-batch in the training of BNNs.

E.7. Takeaways and Perspective

Uncertainty quantification for the predictions of large size neural networks remains an open issue. In this work, we have shown a new way to enable data sub-sampling for Bayesian Neural Network independent from gradient based approximation such as Stochastic Gradient Langevin Dynamic.

First, we have demonstrated that a raw estimation of the likelihood based on a noisy loss introduces a bias in the posterior sampling if not taken into account. We then have shown that a generalization of the Metropolis Hastings algorithm allows us to eliminate the bias and to exactly sample the posterior even with very strong noise. This necessitates an additional "noise penalty" that corresponds to the variance of the noisy loss difference and exponentially suppresses the MCMC acceptance probability.

In practice, the noise penalty corresponds to replacing a single large data set by multiple smaller sub sampled mini batches associated with an uncertainty over their losses. We have shown how to interpret this term as a regularization. Varying the size of the mini-batches enables a natural calibration that we have compared to other techniques such as tempered Safe Bayes approaches.

Based on this calibration principle, we have provided a benchmark that empirically showed good predictive performances of PBNNs. We hope that combining data sub-sampling with other Monte Carlo acceleration techniques such as HMC could allow to compute uncertainties for model sizes not reachable until now.

Lastly, PBNN could be particularly suited in the case when the data sets $\mathscr{D}$ are distributed across multiple decentralized devices as in the typical federated learning setup. Indeed, the noise penalty is determined by the variance of the losses computed on each individual data set. In principle, PBNN should enable the possibility to compute uncertainty with separate data sets without exchanging them.

Figure E.4: 4 examples of Air Liquide demand forecast prediction. Left figures correspond to vanilla BNN with batched loss and right figures correspond to PBNN.

Chapter F

Influence Function for time series

Disclaimer: This chapter is a Work In Progress. A final version should be delivered by the end of December.

Foreword

Some sections of this chapter are either verbatim copies or adapted from existing Confiance's reports.

TODO: add acronym: RNN, IF, TS

Contents

F.1. Motivation

Influence Function (IF) has already proved its potential for Dataset Exploration and Cleaning in Picard et al. [2022] and references therein. However, very few works on IF really focused on Time Series (TS) and fewer looks at its application to sequential-based models such as Recurrent Neural Networks (RNNs). Inspired by the work of Alaa and van der Schaar [2020], which leverages IF properties to estimate predictive uncertainty of an RNN model, we wanted to challenge the DEEL's library Influenciae in order to apply their approaches on a Confiance's Use Case: Air Liquide EMS dataset.

F.2. Influence Function for Time Series

F.2.1 Introduction to Influence Function

[1]

Based on the work of Hampel [1974] we will formulate the theory of existence of the ıinfluence function (IF) of statistical functionals, and we will later describe how to define it for parametric neural network models. Let Ω be a complete separable metric space, let T be a vector-valued mapping from a subset of the probability measures on Ω into the k-dimensional Euclidean space $\mathbb{R}^k$, and let F lie in the domain of T. Let δ_ω denote the atomic probability measure concentrated in any given $\omega \in \Omega$. Then the vector-valued influence function of T at F is defined pointwise by:

$$IF_{T,F}(\omega) = \lim_{\varepsilon \to 0} \frac{T[(1-\varepsilon)F + \varepsilon\delta_\omega] - T(F)}{\varepsilon} \tag{F.1}$$

if this limit exists for every point in $\omega \in \Omega$

The intuition here is that we focus on the optimal estimator T changes, T being a M-estimator, when the distribution it is trained on changes in the direction of δ_ω. In practice, these changes can be the addition or subtraction of a given data-point or group of data-points.

In their seminal work Cook and Weisberg [1980] are one of the first to look at its practical applications. In particular, they compute the influence function of linear regression models empirically, that is, by seeing how the linear coefficients change when a point is taken out of the training set. They use this information to determine which data-points are the most influential. To better analyze this "influence", they define the famous Cook's distance metric that can be defined as follows:

$$D_i(X^T X, ps^2) = \frac{(\hat{\beta}_{(i)} - \hat{\beta})^T (X^T X)(\hat{\beta}_{(i)} - \hat{\beta})}{ps^2} \tag{F.2}$$

[1] This introduction was adapted from the deliverable ISX-EC5-L5.2.4.1-Anomaly_Detection_on_Images available at https://irtsystemx.sharepoint.com/:b:/s/IAdeConfiance833/ESUwps90pfNNpUXb1iMWXvEBrrqT6Lsx7yKdTa9mSFfQjw?e=hVJi1E

where X is the data matrix, $\hat{\beta}$ the coefficients of the linear model trained on the whole dataset, and $\hat{\beta}_{(i)}$ those of the model trained on the dataset with the i-th data-point taken out.

Moving forward in time to 2017, Koh & Liang publish their first paper on the concept of influence function applied to neural network models. In Koh and Liang [2017], they propose a mathematical formulation that is easily applicable to this kind of model thanks to their built-in ability to compute gradients on modern auto-differentiation ML frameworks. Essentially, for a group of training points $z_1, ..., z_n$ where $z_i = (x_i, y_i) \in \mathscr{X} \times \mathscr{Y}$, a set of parameters $\theta \in \Theta$, a loss function $l(z, \theta)$ and an empirical risk minimizer $\hat{\theta} = arg\,min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} l(z_i, \theta)$, they posit that the change in model weights induced by infinitesimally up-weighting a data-point is equal to:

$$\mathscr{I}_{up,loss}(z, z_{test}) = -\nabla_\theta l(z_{test}, \theta)^T H_{\hat{\theta}}^{-1} \nabla_\theta l(z, \hat{\theta}) \tag{F.3}$$

where $H_{\hat{\theta}} = \frac{1}{n} \sum_{i=1}^{n} \nabla^2 l(z_i, \hat{\theta})$ is the Hessian and is assumed to be positive definite.

They apply this technique to the comprehension of the model's predictions, the detection of potential outliers and the fixing of mislabeled data-points.

As a potential limitation, the authors raise the issue of the computational challenge of calculating and inverting the Hessian with respect to the parameters, a problem that grows quadratically with the amount of the model's parameters. To solve this problem, they propose two solutions in their paper: the direct computation of the inverse Hessian-vector products through conjugate gradient descent, and the stochastic estimation of the Hessian through sampling and through the first Taylor expansion of the inverse.

F.2.2 Influence Function & RNN

Recurrent Neural Networks (RNNs) are sequence-based models which have been proven to be effective in a wide-range of domain applications involving temporal sequences such as: stock market predictions Bao et al. [2017], service demand forecasting Wen et al. [2017] or reinforcement learning Wang et al. [2018]. As for any Machine Learning system estimating (reliably) predictive uncertainty of RNNs is of utmost importance as they can be used in high-stakes applications. Even though various research works were conducted for uncertainty estimation in feed-forward networks (e.g. Lakshminarayanan et al. [2017]), equivalent methods for RNN models are still lacking.

In this context, the work of Alaa and van der Schaar [2020] particularly stand out, from our point of view, as they leveraged IF in order to get an estimate of prediction uncertainty for RNN models. In order to do so, they adopted a frequentist approach that is to create multiple perturbed version of a trained RNN model and collect the outputs of these perturbed copies. Confidence intervals are then built based on the jackknife residuals and the variability of these resampled outputs. As it can be noticed, the procedure is entirely *post-hoc*, hence it neither interferes with model training nor compromises its accuracy.

Since RNNs rely on sequences, the generation of perturbed versions operates on "blocks" of dependent data point, in our context entire time series, instead of individual observations. Thus, they used a *blockwise influence function* $\mathscr{I}(.)$, a generalization of conventional IF, to estimate the parameters of the model if a specific data block was removed of the training dataset through the relationship in F.4:

$$\mathscr{I}(DataBlock) \approx \hat{\theta}_{RNN}(\mathscr{D}) - \hat{\theta}_{RNN}(\mathscr{D} \backslash DataBlock) \tag{F.4}$$

where $\mathscr{D}$ denotes the training dataset effectively used for training the original RNN model parameterized by $\hat{\theta}_{RNN}$.

The overall process could be sum up as follows:

(Step 1) Train a RNN model of parameters $\hat{\theta}_{RNN}$ on $\mathscr{D}$

(Step 2) Sample *DataBlock*s from $\mathscr{D}$

(Step 3) Compute the influence vector: IF by applying $\mathscr{I}$ on each sampled block

(Step 4) Using F.4 estimate $\hat{\theta}_{RNN}(\mathscr{D} \backslash DataBlock)$ for each block

(Step 5) For each perturbed model get the perturbed predictions: it will constitute the prediction variability set: $\mathscr{V}$

(Step 6) From $\mathscr{V}$ build Confidence Intervals

F.3. Experiments

F.3.1 UC: Air Liquide, modified EMS

In order to challenge the approach proposed by Alaa and van der Schaar [2020] we were looking for a Use Case where RNN models could be an interesting solution. Naturally, the Air Liquide EMS dataset appeared to be a relevant use case. However, it is composed of a large amount of raw sensor data for different types of physical measurements (e.g. pressure, temperature) acquired at a varying sampling rate over the course of a full year. As training the model is not the main objective here we decided to focus on a subset built by Olivier Antoni and Marielle Malfante from the CEA in their previous work [2] that can be found in the Confiance's wiki.

From the raw data, they first proceed to a resampling of the data using linear interpolation to achieve a rate of one sample per minute. Then, they focus on time series of sensors whose operating regime is rather stable in normal operation. This is especially the case with certain temperature sensors for which the signal in normal operation consists of a main pattern, which is periodically repeated over time, being slightly deformed. For these temperature sensors, the period of the signal is approximately 7 hours. More specifically, a subset was built considering only the *System_3-TI1223.PV* temperature sensor of the Air Liquide dataset.

[2] Parts of this subsection are directly copied or adapted from their section in the report ISX-EC5-LIV-1513-L5.2.4.1-Anomaly_Detection_in_Time_Series__Promising_Tools

Consequently, we have a regularly sampled and univariate Time Series which will be divided into smaller Time Series. In all the experiments, independently of the window splitting of Time Series, data from the month of January will be used to build the training set, data from February will constitute the validation set and March will constitute the test set.

F.3.2 RNN single-output

Model Architecture

Window splitting & Data normalization

Training procedure

Model performance analysis

Computing predictions intervals

F.3.3 RNN multi-output

Model Architecture

Window splitting & Data normalization

Training procedure

Model performance analysis

Computing predictions intervals

F.4. Results

F.4.1 RNN single-output

Some Examples

Statistics on Interval Length

F.4.2 RNN multi-output

Some Examples

Statistics on Interval Length

F.4.3 Limitations

Background work

Computation Power

F.5. Conclusions

Chapter G

Lipschitz networks for robustness by-design

G.1. Introduction

Deep neural networks are known to be vulnerable to adversarial attacks: an imperceptible perturbation in the input can yield a large change in the output and thus misclassify the input. This sensitivity can be measured by the Lipschitz constant of the network, defined as the smallest value L for which

$$\forall x_1, x_2,\ \|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\|. \tag{G.1}$$

A natural way to make networks more robust is to constrain the Lipschitz constant of the network. This constraint can be enforced "by-design" in the layers of the network: such networks are called Lipschitz networks.

The works presented here are the continuation of an action sheet in batch 1. Only the developments done in batch 2 are presented in this chapter. The reader should refer to the previous Confiance EC4 report for a comprehensive description of Lipschitz networks and guidelines on how to build and train these networks using the deel-lip library.

This chapter consists in three main contributions:

- **New losses for 1-Lipschitz networks** New losses are introduced to handle the accuracy-robustness trade-off in binary and multi-class problems. These losses are compared on CIFAR-10 and Renault Welding UC.
- **Image semantic segmentation** Lipschitz networks are used for robust segmentation, which is a totally new task for this kind of networks. Two datasets are used: a toy public dataset (Oxford IIIT Pets) and the Valeo Woodscape UC.
- **Orthogonal convolutions** A new way to encourage orthogonality in convolutions is presented. It is based on the use of a regularization term added to the main loss. This method was proposed by the DEEL team.

All the experiments in the sections below are reproducible with the available codes in the Gitlab repositories: https://git.irt-systemx.fr/confianceai/ec_4/as11_uc and https://git.irt-systemx.fr/confianceai/ec_4/as21_lipschitz. All models were built and trained using the open-source deel-lip library: https://github.com/deel-ai/deel-lip. The deel-lip package, developed by the DEEL team, is maintained and frequently upgraded to reflect the developments in Confiance.ai program.

G.2. New losses for 1-Lipschitz networks

1-Lipschitz networks handle a trade-off between accuracy and robustness. The loss is the keystone: a hyper-parameter in the loss controls this trade-off. This section introduces insights on the loss hyper-parameter and presents new losses for 1-Lipschitz in multi-class problems.

G.2.1 Considerations on losses for Lipschitz networks

Let's consider a standard neural network f defined by $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$, with input $\boldsymbol{x}$, weights $\boldsymbol{\theta}$ and output logits $\boldsymbol{y}$. A classification loss $\mathscr{L}(\boldsymbol{y}, \boldsymbol{t})$, with label targets $\boldsymbol{t}$, is minimized w.r.t the weights $\boldsymbol{\theta}$ and we denote the optimal network $\hat{f}_L(\boldsymbol{x})$. The subscript L is the Lipschitz constant of the trained neural network. In practice, for unconstrained networks, this Lipschitz constant L is usually very large.

Dividing naively the optimal network $\hat{f}_L$ by L gives a 1-Lipschitz network: $\hat{f}_1 = \frac{1}{L}\hat{f}_L$. Note that this new network is not more robust than $\hat{f}_L$; indeed the output logits $\frac{1}{L}\boldsymbol{y}$ are very close to zero, and do not ensure that a small perturbation would not change the classification. We can ask what loss must be chosen to get the same optimum $\hat{f}_L$ if we directly train a constrained 1-Lipschitz network? Transferring the Lipschitz constant L from the model to the loss, the solution is straightforward:

$$\begin{aligned}\mathscr{L}(\boldsymbol{y}, \boldsymbol{t}) &= \mathscr{L}(\hat{f}_L(\boldsymbol{x}), \boldsymbol{t}) \\ &= \mathscr{L}(L\hat{f}_1(\boldsymbol{x}), \boldsymbol{t}) \\ &= \mathscr{L}_L(\hat{f}_1(\boldsymbol{x}), \boldsymbol{t})\end{aligned}$$

Roughly speaking, the 1-Lipschitz $\hat{f}_1$ trained with $\mathscr{L}_L$ gives the same classification results as the L-Lipschitz network trained with $\mathscr{L}$. In practice, the loss $\mathscr{L}_L$ has a hyper-parameter L, having a similar behaviour as the Lipschitz constant of the network. Choosing a high L leads to a 1-Lipschitz network with the behaviour of a standard network, i.e. accurate but not robust. A small L reduces the accuracy but improves robustness. Finally, the hyper-parameter L tunes the trade-off between accuracy and robustness. More details on the importance of the loss for 1-Lipschitz networks are given in Béthune et al. [2021]. The losses presented below will highlight this trade-off.

G.2.2 Different hinge losses for multi-class problems

The robust hinge loss for binary classification is simply defined as

$$\mathscr{L}_{\text{hinge}} = (m - y \cdot t)^+ \tag{G.2}$$

with logit y, class target t (being +1 or -1), $x^+ = \max(x, 0)$ and margin $m \in \mathbb{R}^+$ to handle the trade-off between accuracy and robustness. A loss with a large margin m tends to produce high logits, far from the decision boundary, and thus improves robustness.

Extending the binary case to multi-class, we can distinguish two approaches that we call one-vs-one and one-vs-all approaches. Both approaches can be derived in a standard version or a zero-centered version. We will thus describe 4 possible configurations combining 1-vs-1 or 1-vs-all approach with zero-centered or uncentered version. We also design a new formulation, called weighted hinge loss, which is a generalization of the two 1-vs-1 and 1-vs-all approaches. Remember that all equations below are written for a single element (input). The loss for a whole batch is the average of this element-wise term for all the inputs in the batch.

G.2.2.1 One-vs-one version

The one-vs-one formulation for N_c classes is defined as follows:

$$\mathscr{L}_{\text{1-vs-1}} = \frac{1}{N_c} \sum_{i \neq c} \Big(m - (y_c - y_i) \Big)^+ \tag{G.3}$$

The term $(y_c - y_i)$ is the difference between the predictions for the true class c and another class i. These differences are encouraged to be larger than the margin m. This version is the one defined in PyTorch `MultiMarginLoss` and in deel-lip `MultiMargin`.

The above formulation can be adapted to get zero-centered predictions: the same margin m is incited but the predictions are forced to positive for the target class and negative for others. The zero-centered one-vs-one version in Eq. (G.4) is implemented in deel-lip `MulticlassHinge`.

$$\mathscr{L}_{\text{1-vs-1}}^{\text{centered}} = \frac{1}{N_c} \sum_{i \neq c} \left[\left(\frac{m}{2} - y_c \right)^+ + \left(\frac{m}{2} + y_i \right)^+ \right] \tag{G.4}$$

G.2.2.2 One-vs-all version

Unlike the one-vs-one version expressed as an average on multiple one-vs-one losses, the one-vs-all configuration is based on the maximum prediction among all other predictions y_i:

$$\mathscr{L}_{\text{1-vs-all}} = \Big(m - (y_c - \max_{i \neq c} y_i) \Big)^+ \tag{G.5}$$

The term $(y_c - \max_{i \neq c} y_i)$ is the difference between the prediction for the true class c and the largest prediction for other classes $i \neq c$. Unlike the one-vs-one approach, only the smallest difference $(y_c - y_i)$ will be taken into account to compute the loss. This one-vs-all version is the one provided in TensorFlow/Keras `CategoricalHinge` and also in deel-lip `CategoricalHinge`.

We can also adapt this loss for a zero-centered version:

$$\mathscr{L}_{\text{1-vs-all}}^{\text{centered}} = \left(\frac{m}{2} - y_c\right)^+ + \left(\frac{m}{2} + \max_{i \neq c} y_i\right)^+ \tag{G.6}$$

G.2.2.3 Weighted-hinge version

The weighted-hinge loss is a generalized formulation of the two losses presented above. A weighting term based on softmax is applied to each hinge term. A temperature factor τ is introduced in the softmax to slide between the 1-vs-1 and the 1-vs-all versions. The weighted hinge term for a single element is defined as follows:

$$\mathscr{L}_{\text{weighted-hinge}} = \sum_{i \neq c} w_i \left(m - (y_c - y_i)\right)^+ \quad \text{with } w_i = \frac{e^{\tau y_i}}{\sum\limits_{j \neq c} e^{\tau y_j}} \tag{G.7}$$

The weights w_i are the softmax of the logits except for the logit from the true class c, i.e. the logits $i \neq c$. Since the weights result from a softmax operation, the sum of these $(N_c - 1)$ weights equals to 1. Note that:

- a temperature $\tau = 0$ gives weights all equal to $w_i = \frac{1}{N_c - 1}$. This is equivalent to the 1-vs-1 formulation, ignoring a multiplication factor $\frac{N_c}{N_c - 1}$.
- a temperature $\tau \to +\infty$ is strictly equivalent to the 1-vs-all version: $w_i = 1$ for the neuron having the highest logit ($i = \underset{j \neq c}{\text{argmax}}\, y_j$) and $w_i = 0$ for all other neurons.

Like the formulations above, the weighted-hinge loss has a centered version:

$$\mathscr{L}_{\text{weighted-hinge}}^{\text{centered}} = \sum_{i \neq c} w_i \left[\left(\frac{m}{2} - y_c\right)^+ + \left(\frac{m}{2} + y_i\right)^+\right] \tag{G.8}$$

G.2.2.4 The squared hinge losses

All the hinge losses formulated above can also be derived in squared hinge versions. In the previous equations, all ReLU terms $(x)^+$ are squared: $(x)^{+^2}$.

G.2.3 Auto margin hinge losses

A drawback of hinge losses, as presented above, is that the margin hyper-parameter is difficult to choose. Moreover, in the multi-class setting, there is no evidence that a single margin should be chosen for all classes. Obviously this will be problematic for large scale datasets such as Imagenet with 1000 classes.

Within the DEEL project (www.deel.ai), a new loss has been proposed and published in Serrurier et al. [2022] to automatically adjust the margins with a single hyper-parameter α. The principle will be explained with 1-vs-all version but can be applied with any hinge variant. The loss for a single element of class c with the corresponding margin m_c is

$$\mathscr{L}^{auto}_{\text{1-vs-all}} = \left(m_c - (y_c - \max_{i \neq c} y_i)\right)^+ - \alpha m_c \tag{G.9}$$

Making this margin m_c learnable, it will be updated according to

$$\frac{\delta \mathscr{L}^{auto}_{\text{1-vs-all}}}{\delta m_c} = \begin{cases} 1-\alpha, & \text{if } m_c > y_c - \max_{i \neq c} y_i. \\ -\alpha, & \text{otherwise.} \end{cases} \tag{G.10}$$

Thus, summing over all samples of class c, m_c variable will be increased (resp. decreased) if less (resp. more) than a fraction of α samples are within the margin, i.e $m_c > y_c - \max_{i \neq c} y_i$. The margin m_c will be automatically adjusted such that only α percent of samples are smaller than the margin.

Serrurier et al. [2022] also propose a weighted hinge version with automatic margin:

$$\mathscr{L}^{auto}_{\text{weighted-centered-hinge}} = \sum_{i \neq c} w_i \left[\left(\frac{m}{2} - y_c\right)^+ + \left(\frac{m}{2} + y_i\right)^+\right] - \alpha m_c \quad \text{with } w_i = \frac{e^{\tau y_i}}{\sum_{j \neq c} e^{\tau y_j}} \tag{G.11}$$

With such a loss and classical data augmentation, it is possible to achieve more than 91% of accuracy on CIFAR-10 dataset. This loss and a special train step have been implemented in *DEEL.LIP*[1] library.

G.2.4 Cross-entropy loss for 1-Lipschitz networks

The cross-entropy loss is the most common loss for classification problems. For a single element, it is expressed as

$$\mathscr{L}(\boldsymbol{y}, \boldsymbol{t}) = -\boldsymbol{t} \cdot \log(\sigma(\boldsymbol{y})) \tag{G.12}$$

where $\boldsymbol{y}$ are the N_c logits, $\boldsymbol{t}$ the one-hot encoded targets, and σ is the softmax function. A version for Lipschitz networks can be derived with a temperature hyper-parameter to tune the trade-off:

$$\mathscr{L}_\tau(\boldsymbol{y}, \boldsymbol{t}) = -\boldsymbol{t} \cdot \log(\sigma(\tau \boldsymbol{y})) \tag{G.13}$$

Unlike hinge losses where robustness requires large margins, with cross-entropy: the smaller the temperature, the larger the robustness is.

G.2.5 Other losses for 1-Lipschitz in the literature

In the above subsections, usual losses were expressed with a hyper-parameter to robustify Lipschitz networks, which is new and was formalized in Béthune et al. [2021]. Other techniques were though presented in the literature to deal with the trade-off between accuracy and robustness. Two of them are detailed here.

[1] https://github.com/deel-ai/deel-lip distributed under MIT License (MIT)

G.2.5.1 Lipschitz-margin training

Tsuzuku et al. [2018] proposed to shift the output logits in order to encourage a margin m between the logit y_c of the true class and other logits:

$$\forall i \neq c,\ y_i \leftarrow y_i + m \tag{G.14}$$

Any common loss is then applied on the shifted logits. In their paper, the authors applied the standard cross-entropy. Note that the performance of the trained network is still limited by the standard cross-entropy where no hyper-parameter is used.

G.2.5.2 Certificate regularization

Singla et al. [2021] introduced a regularization term added to the main loss term $\mathscr{L}(\boldsymbol{y})$. This regularization aims at maximizing the difference between the logit of the target class and the highest other logits:

$$\mathscr{L}(\boldsymbol{y},\boldsymbol{t}) = \ell(\boldsymbol{y},\boldsymbol{t}) - \gamma\left(y_c - \max_{i\neq c} y_i\right)^+ \tag{G.15}$$

where γ is the regularization factor. Note that this regularization term can be compared to the KR term in hKR (hinge-Kantorovich-Rubinstein) loss. For more details on hKR loss, see Confiance EC4 report. Both regularization term and KR term tend to move the logit of the true class away from the other logits. In other terms, these terms tend to move the input far from the decision boundary. Like in Tsuzuku et al. [2018], there is no hyper-parameter in the standard loss ℓ chosen by the authors.

G.2.6 Summary table

Table G.1 summarizes the different losses to make Lipschitz networks robust.

Loss	Standard	Robust
Cross-entropy	$-\boldsymbol{t}\cdot\log(\boldsymbol{\sigma}(\boldsymbol{y}))$	$-\boldsymbol{t}\cdot\log(\boldsymbol{\sigma}(\tau\boldsymbol{y}))$
Hinge	$(1-\boldsymbol{t}\cdot\boldsymbol{y})^+$	$(m-\boldsymbol{t}\cdot\boldsymbol{y})^+$
Squared hinge	$(1-\boldsymbol{t}\cdot\boldsymbol{y})^{+^2}$	$(m-\boldsymbol{t}\cdot\boldsymbol{y})^{+^2}$
hKR	N/A	KR + α hinge
Squared hKR	N/A	KR + α squared_hinge
L-margin training	N/A	shifted logits: $y_i \leftarrow y_i + m$
Certificate regul	N/A	$\ell(\boldsymbol{y},\boldsymbol{t}) - \gamma\left(y_c - \max_{i\neq c} y_i\right)^+$

Table G.1: Losses for 1-Lipschitz networks

G.2.7 Experiments on CIFAR-10

Losses for 1-Lipschitz are tested on CIFAR-10 dataset. The performance of networks trained with different losses is measured with the accuracy on the test set. Regarding robustness, it can

be measured with different metrics, like robust accuracy using adversarial attacks or certified accuracy for a given perturbation strength. Here, we measure robustness with the KR value on the test set, which represents how large the output logits are, i.e. how far the elements are from the decision boundary. The larger the KR is, the more robust the network is. Code to reproduce experiments is available in the corresponding Gitlab repo.

Fig. G.1 highlights the trade-off between accuracy and robustness. Each point represents a trained model with a specific loss (defined by the color and shape of the point). Note that hinge losses are presented in their one-vs-one version. The accuracy / robustness trade-off is emphasized by the Pareto front emerging with the points. We notice that some losses never reach this front, meaning that they are not competitive compared to others close to the front. This is the case for all squared hinge losses: centered and uncentered squared hinge, centered and uncentered squared hKR. In contrast, some losses are present on the Pareto front: cross-entropy, centered hinge and centered weighted hinge losses. Depending on the side of the front, one loss is more efficient than the others. For instance, for high accuracy and low robustness, the centered hinge and weighted hinge losses seem better. However, for lower accuracy and higher robustness, the cross-entropy loss gives better results.

Moreover, the uncentered losses (uncentered hinge, squared hinge, hKR and squared hKR) are almost always worse than their centered counterparts. Finally, the weighted hinge, a generalization between one-vs-all and one-vs-one hinge losses, have two hyper-parameters, the hinge margin and the softmax temperature. From these experiments, we cannot conclude on the best choice for temperature: the best networks on the Pareto front do not always have the same temperature.

In conclusion, we encourage to use either cross-entropy or centered one-vs-one hinge loss to maximize the accuracy and robustness performances. The temperature in cross-entropy or the margin in hinge loss must be tweaked for the desired trade-off.

G.3. Application to Renault Welding UC

1-Lipschitz networks have been applied on the Renault Welding UC. Losses for 1-Lipschitz were tested with different values of hyper-parameters, leading to more accurate or more robust networks.

Code is available on the corresponding Gitlab repo. Like our other experiments, trainings are based on the TensorFlow/Keras framework and deel-lip library.

G.3.1 Configuration of the experiments

G.3.1.1 Dataset used

The Renault Welding dataset consists of images of weldings from different industrial parts and multiple cameras, called cordons. The problem can be seen as a binary classification problem on good ("OK") and bad ("RETOUCHE") weldings. Our study focused on a single cordon, c19 (see Fig G.2), containing about 1500 images with 80% of OK weldings and 20% of RETOUCHE weldings. As no split is provided with the dataset, we decided to split the c19 dataset into train/validation/test subsets of respectively 80% / 10% / 10% images. The test set is thus

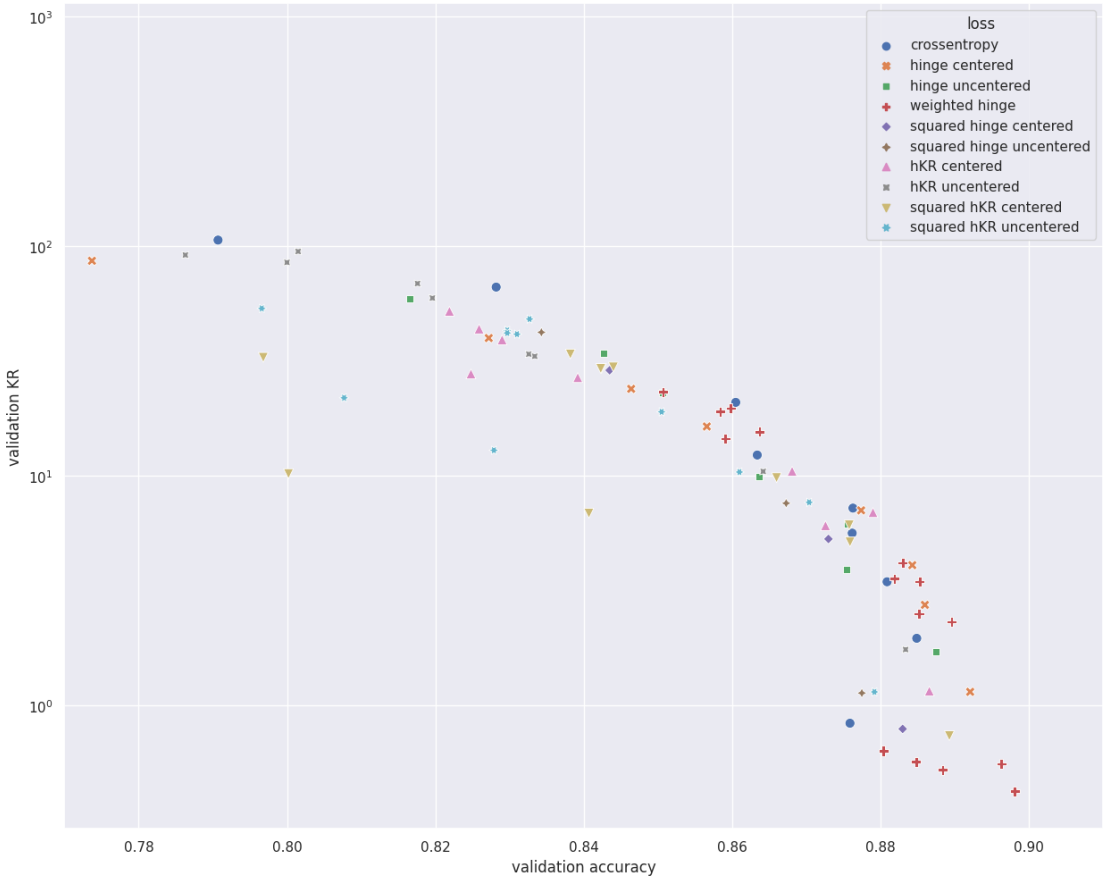


Figure G.1: Accuracy-robustness trade-off for models trained with different losses and their hyper-parameters. The x-axis is the accuracy on the validation set and the y-axis is the validation KR metric, representing the robustness of the network.

composed of 121 OK images and 30 RETOUCHE images. For reproducibility and comparison purposes, these splits are provided in CSV files in the Gitlab repo.

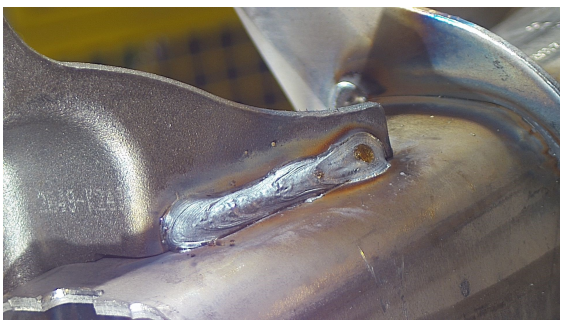


Figure G.2: Example of an OK welding image from cordon c19 in Renault Welding dataset.

Since the dataset is heavily imbalanced, the train set is equalized by duplicating RETOUCHE images multiple times in order to have the same number of OK and RETOUCHE images in the train set. A CSV file with a balanced train set is also available.

Finally, the images are pre-processed: resized to 224x224 and rescaled to float values between 0 and 1.

G.3.1.2 Training configuration

We trained a small VGG-like model with 9 convolutional layers and 2 fully-connected layers:

C32 - PL - C64 - PL - C128 - PL - C128 - C128 - PL - C256 - C256 - PL - C512 - C512 - GPL - FC256 - FC1

with Cxx for a convolutional layer with xx filters, FCxx a fully-connected layer with xx neurons, PL a 2x2 pooling layer and GPL a global pooling layer. Convolutional and fully-connected layers are followed by an activation layer.

During training, the train set is augmented with simple random operations: brightness, contrast, saturation, Hue offset, zoom in. Note that only a few augmentations were used, so the trained network is not designed to be robust against specific perturbations, such as rotations, blur or dead pixels. But if required, data augmentation can be improved to deal with such alterations.

The Adam optimizer is used with a cosine decay learning rate. Please refer to the report of batch 1 for guidelines on how to build and train 1-Lipschitz networks.

G.3.1.3 Tested losses for 1-Lipschitz

The losses for 1-Lipschitz presented in the previous section were tested on this use case to handle the trade-off between accuracy and robustness. Results for only some losses are given in Sec. G.3.3: hinge, hKR and cross-entropy. The results for other losses are similar, but not presented to ease the readability.

G.3.2 Performance metrics

The performance of networks is measured with the F_1 score on the test set. F_1 metric was chosen because it is better suited than accuracy for imbalanced datasets.

To measure robustness, the networks are attacked with adversarial methods, such as L2-PGD or DeepFool: for a given perturbation budget ε, the attacker tries to find the best adversarial example satisfying the budget constraint. The F_1 score is then measured using the obtained adversarial images for a given budget ε; it is called robust F_1 score. The more robust the network is, the smaller the gap is between robust and clean F_1 scores.

Attacks are run for multiple perturbation budgets ε in order to see the evolution of the robust F_1 score as a function of ε.

G.3.3 Results

G.3.3.1 Robust F_1 scores under adversarial attacks

Fig. G.3 provides an understanding about the effect of loss hyper-parameter. Each line represents the robust F_1 score under DeepFool attack for a given trained network:

- a standard network (gray line) is accurate (high F_1 score when no attack, i.e. $\varepsilon = 0$). However, the score dramatically drops when attacking the network, even for small perturbation strength (i.e. small ε).
- a 1-Lipschitz network was trained with a robust hinge loss and a margin $m = 0.5$ (blue line). The clean score is very high, as high as for the standard network. Moreover, the network is more robust than the standard network.
- a 1-Lipschitz network was trained with an even more robust hinge loss using $m = 5$ (red line). The network is very robust to perturbation, but with a lower clean performance. It illustrates the trade-off between accuracy and robustness.

G.3.3.2 ROC curves

Another visualization for binary classification is the ROC curve (*Receiver Operating Characteristic*) and the corresponding AUC (*Area Under Curve* which is also a good metric to measure performance -but not for robustness-). Fig. G.4 plots the ROC curves for the different models. The standard model shows the best AUROC, which reflects the F_1 score. However, depending on if we prefer to emphasize the precision or recall score, the robust models can achieve precision or recall by changing the decision boundary (red dots in ROC represent the default decision boundary set to zero). Note that modifying the decision boundary to focus on either precision or recall will modify the success of adversarial attacks and change the robust performance of the models.

G.3.3.3 Adversarial perturbations

Finally, Fig. G.5 exhibits the adversarial perturbation, obtained using DeepFool, that changes the classification. We compare the perturbation for the standard network and for a robust 1-Lipschitz network. The figure clearly highlights that the adversarial perturbations for a Lipschitz network are structured and condensed on the welding. In contrast, the perturbations for the standard network are totally noisy without any structure.

Moreover, the L_2 norm of the adversarial perturbation (value above each perturbation image) is much larger for 1-Lipschitz networks than for standard networks, confirming the higher robustness of Lipschitz models.

G.4. Robust segmentation with 1-Lipschitz networks

There is no mention of Lipschitz networks applied to semantic segmentation in the literature. But this is a natural extension of our works on image classification. This section presents our

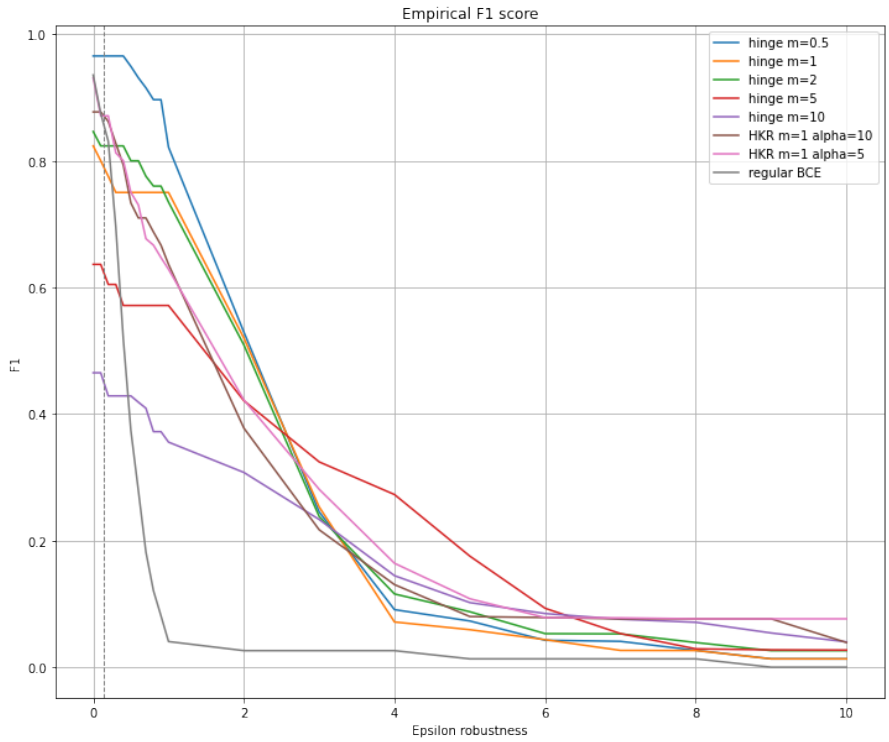


Figure G.3: Robust F_1 score as a function of the attack perturbation budget ϵ. The three curves correspond to three trained networks. The blue network is a standard network with no Lipschitz constraint. The orange one is a 1-Lipschitz network trained with a robust hinge loss $m = 0.5$. The purple network is a 1-Lipschitz network trained with an even more robust hinge loss $m = 5$.

preliminary works applied on a toy public dataset and on Valeo Woodscape UC.

G.4.1 Network architecture

The task of image semantic segmentation is to predict a class for every pixel of the input image. The first known NN architectures for this specific task were proposed in 2015 and several more complex and more efficient networks have been proposed ever since. Here is a non-exhaustive list of the most famous architectures:

- Fully Convolutional Network (FCN) Long et al. [2015]
- U-Net Ronneberger et al. [2015]
- Pyramid Scene Parsing Network (PSPNet) Zhao et al. [2017]
- DeepLab (Chen et al. [2017]) and all its versions. DeepLab v3+ is one of the most used architectures nowadays.

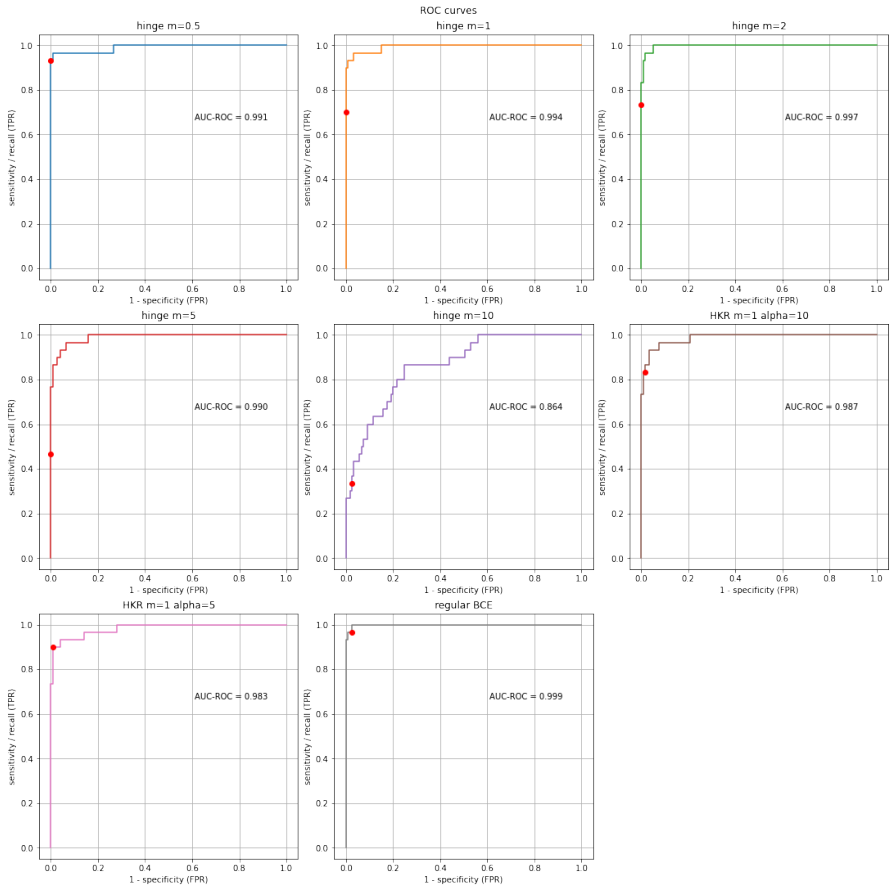


Figure G.4: ROC curves and their corresponding AUROC scores for the trained models. The last one is the standard model without Lipschitz constraints. Red dots represent the default decision boundary set to zero. If we want to focus on either precision or recall, the decision boundary can be moved along the curve and would change the F_1 score as well as the robustness of the network.

The common strategy in these architectures is to stack an encoder part to extract features and a decoder network to propagate the features in the spatial domain. To improve spatial resolution, they collect information at different resolutions. Each architecture has its own way of collecting spatial information through multiple resolutions: skip connections, atrous convolutions, spatial pyramid pooling, etc.

In this study, we first focus on the simplest architectures, FCN and U-Net, to build Lipschitz counterparts. The FCN architecture proposed by Long et al. [2015] is the most simple one: an encoder network (VGG16 in the original paper) reduces the spatial dimensions with a factor of 32 (e.g. from size 224x224 to 7x7). A 32x upsampling layer is appended to recover the input shape. This architecture is called FCN-32s. To improve spatial resolution of the prediction, FCN-16s and FCN-8s architectures combine predictions at different intermediate layers (i.e. at

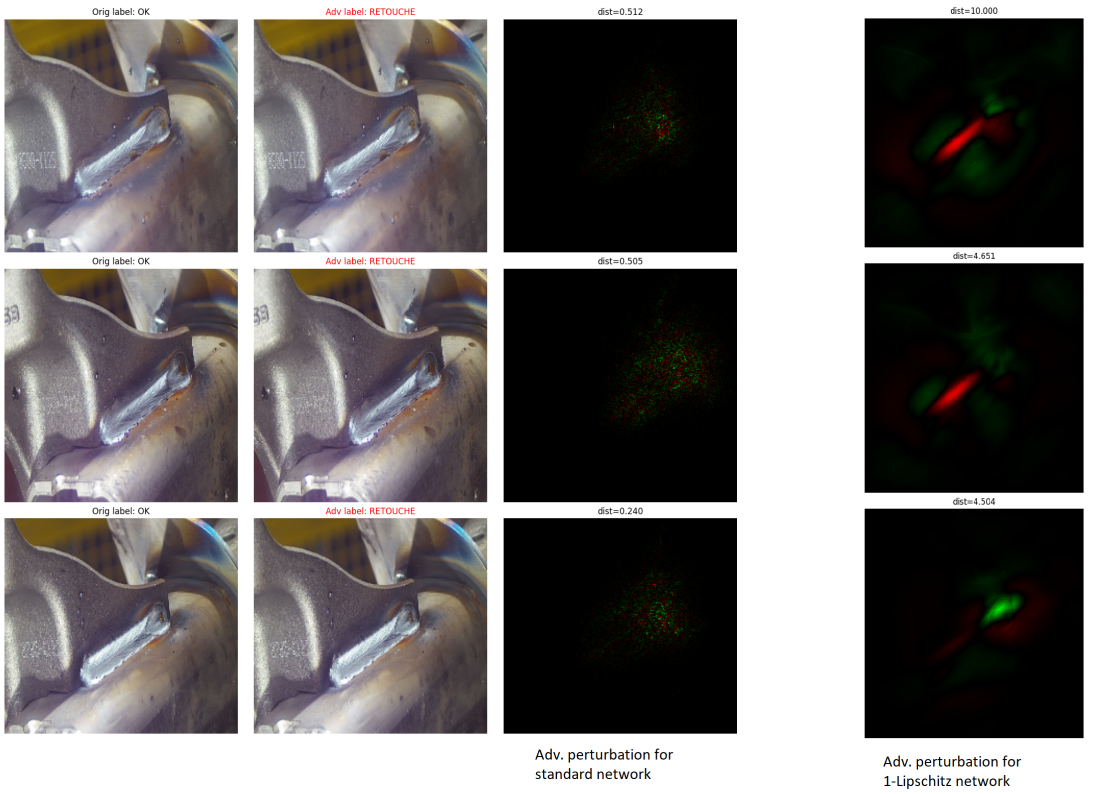


Figure G.5: Adversarial perturbations obtained by DeepFool attack on the standard network and a robust 1-Lipschitz network.

higher resolutions) to provide finer details. Fig. G.6 presents these FCN architectures. The U-Net in Ronneberger et al. [2015] is very similar to FCN but proposes to concatenate predictions in the decoder using all intermediate layers of the encoder.

The deel-lip library was originally not suited to build such networks because some Lipschitz layers were not available, especially the transposed convolutional layer. These new features have been implemented in the latest version (as of December 2022). More complex architectures were not considered but they could be studied in future works.

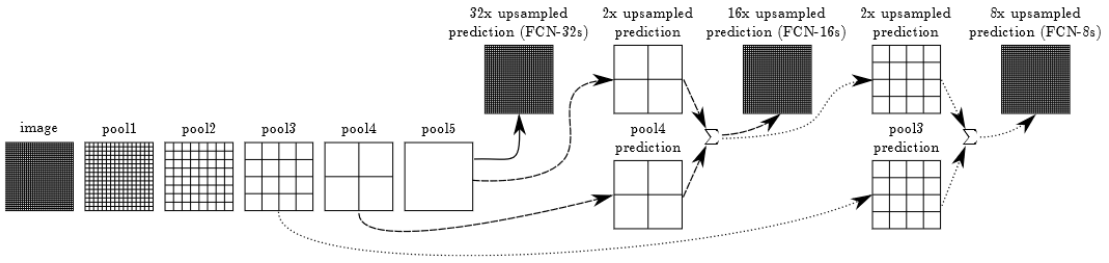


Figure G.6: Fully Convolutional Network (FCN) architecture, from Long et al. [2015]

G.4.2 Toy dataset: Oxford IIIT Pets

Before tackling real use cases, we decided to work with a toy public dataset called Oxford-IIIT Pet Dataset. The dataset contains images of pets (cats and dogs) with trimap annotations: foreground, background and border (a blended zone where the foreground and background are not well-defined). The first two columns in Fig. G.7 display some examples of the dataset.

G.4.2.1 Robust training

We trained a Fully Convolutional Network (FCN) with a small VGG-like encoder. The robust cross-entropy loss was used with various temperatures: lower temperature means more robust networks. Other losses are sometimes used for segmentation, like focal loss or Dice loss. But the cross-entropy loss remains the most frequent.

The mIoU score is a common metric to evaluate semantic segmentation tasks. It measures the intersection over union (IoU) between the ground truth mask and the prediction for each class, then averages across the classes.

Finally, like classification, the robustness is measured under adversarial attacks. We implemented the L2-PGD attack for segmentation. We then evaluate the network with the obtained adversarial images to measure the robust mIoU.

G.4.2.2 Results

For a qualitative visualization of robustness, we first show the prediction maps of six images and their adversarial images after the attack. These segmentation maps for a standard (non-robust) network are presented in Fig. G.7 (columns 3 and 4). The same maps are presented for a robust 1-Lipschitz network (columns 5 and 6). The same conclusions as for classification can be drawn: even if the robust network predicts segmentation maps at a slightly lower quality, the predictions on the adversarial images are far better. This is especially visible on the second-last image (the spotty kitten) where the attack on the standard network can make the cat disappear in the segmentation map. However, the prediction for the robust 1-Lipschitz network is almost unaltered.

G.4.3 Valeo Woodscape UC

The 1-Lipschitz FCN architecture was applied on the Valeo Woodscape use case. This UC contains urban scene images from four fish-eye cameras surrounding the car. The segmentation annotations contain ten categories of objects: void, road, lanes, curbs, rider, person, vehicles, bicycle, motorcycle and traffic sign.

G.4.3.1 Robust training

We trained the same network as in Sec. G.4.2, a FCN network with a small VGG-like encoder. As the architecture is rather small, the performance of the network is not expected to be as competitive as the DeepLab v3+ network provided by Valeo. The objective is rather to compare the performance between the standard network and its same 1-Lipschitz architecture.

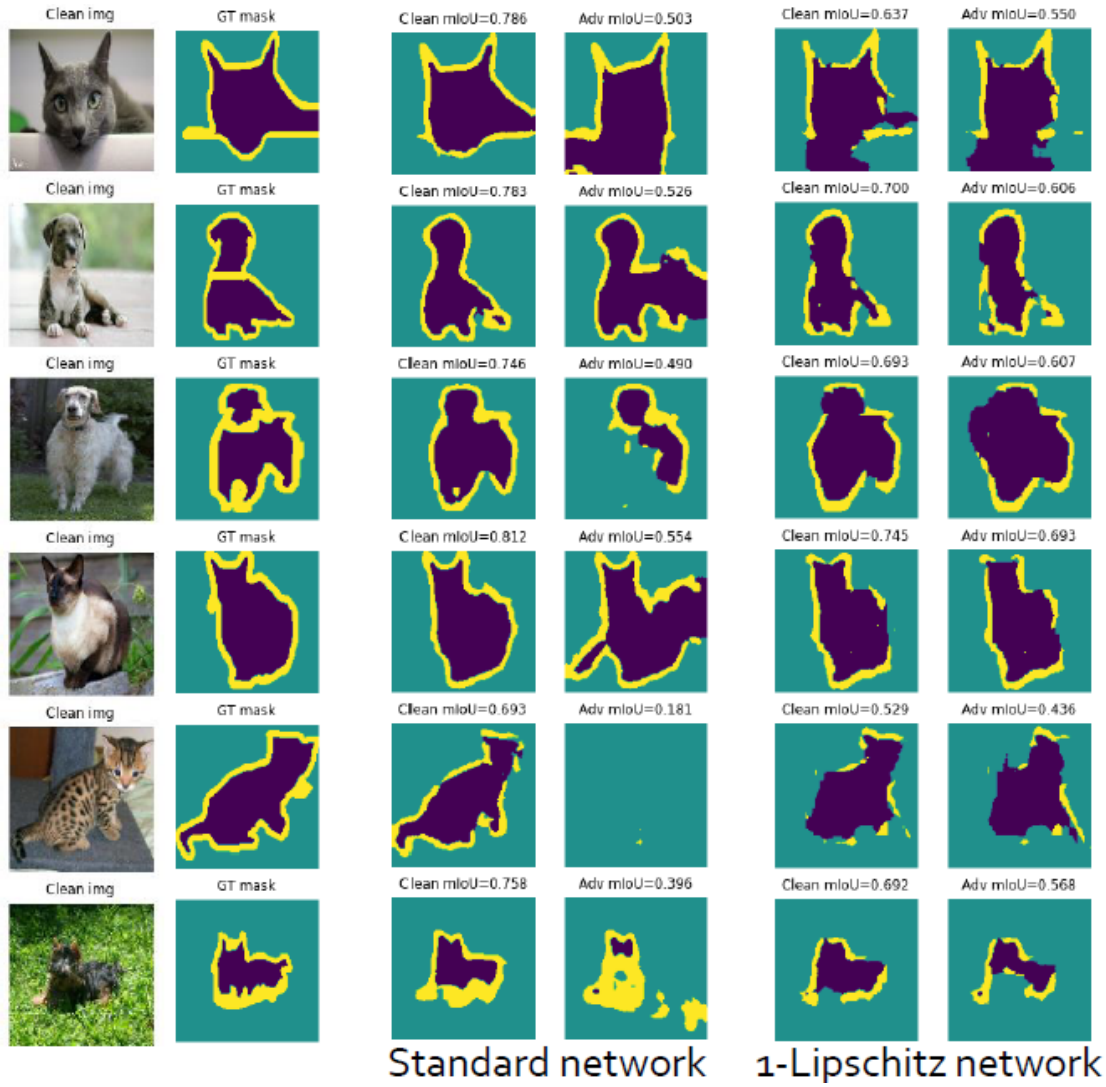


Figure G.7: Standard and robust segmentation maps on Oxford IIIT Pets. The first two columns show six images from the test set and their ground truth segmentation maps. The third and fourth columns are the prediction maps of the original and adversarial images for a standard network. The two last columns are the prediction maps of the original and adversarial images for a robust 1-Lipschitz network.

The training set is slightly augmented to improve generalization (brightness, contrast, saturation, flip left/right). A standard cross-entropy loss is used to train the standard network and a cross-entropy loss with temperature is applied for the Lipschitz model. The networks are trained for 300 epochs with a cosine learning rate scheduler.

The trained models are then attacked with L2-PGD and a perturbation budget of 1. We decided to apply targeted attacks, i.e attacks where we want to change classification towards a

specific class. The targeted class is "void": the attack tries to change predictions of all pixels to be "void" (background). In other words, we want to make objects disappear, such as vehicles, pedestrians or even roads.

G.4.3.2 Results

Fig. G.8 presents our first results of training and attacking a robust Lipschitz network on Valeo Woodscape UC. For both the standard and the 1-Lipschitz networks, we show the prediction maps for the original image and the adversarial image. Values above each segmentation map give the mIoU of the current prediction compared to the ground truth.

The trade-off between accuracy and robustness is emphasized once again. The standard networks exhibit better clean mIoU scores than the robust Lipschitz network: for the 10 randomly chosen images, the standard network yields better scores. However, under attack, the performance of the standard network drops, whereas the 1-Lipschitz network is more robust to the attack: here the robust mIoU is better for the Lipschitz network on 8 images out of 10. Table G.2 below summarizes the clean and robust mIoU on the 10 images for both standard and 1-Lipschitz networks.

Qualitatively, the attack is very efficient on the standard network. The objects, such as roads or vehicles, are "replaced" by the void category. In contrast, the Lipschitz network is almost not altered by the attack.

	Standard	**1-Lipschitz**
Clean mIoU	0.443	0.338
Robust mIoU	0.245	0.334

Table G.2: Clean and robust mIoUs averaged on 10 images of the Valeo Woodscape test set. The mIoU of the 1-Lipschitz network is almost unaltered by the L2-PGD adversarial attack, whereas the standard network is very sensitive to perturbations: the mIoU dramatically drops after attack.

G.5. Orthogonal convolution

In this part we will give a summary of the article "Existence, Stability and Scalability of Orthogonal Convolutional Neural Networks" by Achour et al. [2021] developed within the DEEL project (`www.deel.ai`).

1-Lipschitz layers have interesting properties but do not prevent vanishing gradient: the highest singular value is one, but other singular values can be very small. Using orthogonal layers ensures that all singular values are equal to one, thus preserving gradient norm. Making a fully-connected layer orthogonal is possible by orthogonalizing its weight matrix. However, to the best of our knowledge, the understanding of orthogonal convolutional layers is weak. There is no paper focusing on the theoretical properties of orthogonal convolutional layers. Moreover, most papers only focus on kernel orthogonalization, which cannot imply the orthogonality of the convolutional layer.

Achour et al. [2021] consider the architecture of a convolutional layer as characterized by (M,C,k,S), where M is the number of output channels, C of input channels, convolution kernels are of size $k \times k$ and the stride parameter is S. Thus, applied on input channels of size $SN \times SN$, the M output channels are of size $N \times N$. We denote by $\mathbf{K} \in \mathbb{R}^{M \times C \times k \times k}$ the kernel tensor and by $\mathscr{K} \in \mathbb{R}^{MN^2 \times CS^2N^2}$ the matrix that applies the convolutional layer of architecture (M,C,k,S) to C vectorized channels of size $SN \times SN$.

The paper describes necessary and sufficient conditions for the existence of an orthogonal convolution layer under the circular boundary conditions:

- Row Orthogonality (RO) case, i.e. $M \leq CS^2$: if and only if $M \leq Ck^2$.
- Column Orthogonality (CO) case, i.e. $M \geq CS^2$: if and only if $S \leq k$.

They also denote that orthogonal convolutions are dummies when considering zero-padding, or "valid" boundary conditions.

Besides they rely on a previous work of Wang et al. [2020], which has introduced a regularization method called L_{orth}. Denoting $P = \left\lfloor \frac{k-1}{S} \right\rfloor S$, $L_{orth} : \mathbb{R}^{M \times C \times k \times k} \longrightarrow \mathbb{R}_+$ is defined as follows

- In the RO case, $M \leq CS^2$:

$$L_{orth}(\mathbf{K}) = \|\mathbf{conv}(\mathbf{K}, \mathbf{K}, \text{padding zero} = P, \text{stride} = S) - \mathbf{I}_{r0}\|_F^2 \ . \tag{G.16}$$

- In the CO case, $M \geq CS^2$:

$$L_{orth}(\mathbf{K}) = \|\mathbf{conv}(\mathbf{K}, \mathbf{K}, \text{padding zero} = P, \text{stride} = S) - \mathbf{I}_{r0}\|_F^2 - (M - CS^2) \ .$$

such that

$$\mathscr{K} \text{ orthogonal} \qquad \Longleftrightarrow \qquad L_{orth}(\mathbf{K}) = 0.$$

Since $L_{orth}(\mathbf{K}) = 0$ will never be achieved in practice, Achour et al. [2021] have studied and proved

- **Stability with regard to minimization errors** Does $\mathscr{K}$ still have good 'approximate orthogonality properties' when $L_{orth}(\mathbf{K})$ is small but non zero? Without this guarantee, it could happen that $L_{orth}(\mathbf{K}) = 10^{-9}$ and $\|\mathscr{K}\mathscr{K}^T - Id\|_2 = 10^9$. This would make the regularization with L_{orth} useless, unless the algorithm reaches $L_{orth}(\mathbf{K}) = 0$.
- **Scalability and stability with regard to N:** Remarking that, for a given kernel tensor $\mathbf{K}$, $L_{orth}(\mathbf{K})$ is independent of N but the layer transform matrix $\mathscr{K}$ depends on N: when $L_{orth}(\mathbf{K})$ is small, does $\mathscr{K}$ remain approximately orthogonal and isometric when N grows? If so, the regularization with L_{orth} remains efficient even for very large N.
- **Optimization:** Does the landscape of L_{orth} lend itself to global optimization?

The regularization term $L_{orth}(\mathbf{K})$ to enforce orthogonality in convolutional layers was introduced in the deel-lip library as an alternative to the legacy 1-Lipschitz `SpectralConv2D` layer.

G.6. Conclusions and perspectives

For a summary of advantages and shortcomings of 1-Lipschitz networks, the reader can be referred to the Confiance EC4 report of batch 1.

G.6.0.1 New losses for 1-Lipschitz networks

The loss is the key part for handling the trade-off between accuracy and robustness. Our works aimed at deriving new losses for 1-Lipschitz from standard ones and at improving the hKR loss. The loss hyper-parameters allow tuning the expected robustness level. The results on CIFAR-10 and on Renault Welding UC confirm the good performance of 1-Lipschitz networks in terms of accuracy and robustness, compared to standard networks. Some of these new losses were integrated into the deel-lip library, especially `TauCategoricalCrossentropy`, weighted hinge in `MulticlassHinge` and `MulticlassHKR`, and one-vs-all hinge in `CategoricalHinge`.

G.6.0.2 Robust semantic segmentation

1-Lipschitz networks were successfully applied to image segmentation. The results on the Oxford-IIIT Pet dataset are impressive with good performance under adversarial attacks compared to standard networks. A preliminary study was run on Valeo Woodscape UC with promising results for 1-Lipschitz networks. Adversarial attacks on the standard network succeed in removing objects from the segmentation map, whereas the predictions of the Lipschitz network are almost not altered. These preliminary results should be confirmed with more complex architectures and new losses for 1-Lipschitz based on focal loss.

G.6.0.3 Orthogonal convolution

The works about orthogonal convolution done in the DEEL program, implemented in deel-lip library and were used in Confiance.ai. This transfer from academic context to library development enhances the possibilities for an end-user to build more powerful networks by relaxing the orthogonalization constraint and reducing the memory use.

G.6.0.4 Perspectives

Future works can be categorized in different axes:

- Applications: 1-Lipschitz networks can be applied on new tasks. After consolidating our works on segmentation, we plan to extend Lipschitz networks to object detection.
- Theoretical works: some academic works are ongoing, such as the estimation of the Lipschitz constant with the power iteration algorithm for convolutional layers.
- Links with other works within Confiance.ai: 1-Lipschitz networks can be used and confronted to other topics studied in Confiance.ai program, e.g. explainability, conformal prediction, environmental alterations, etc.

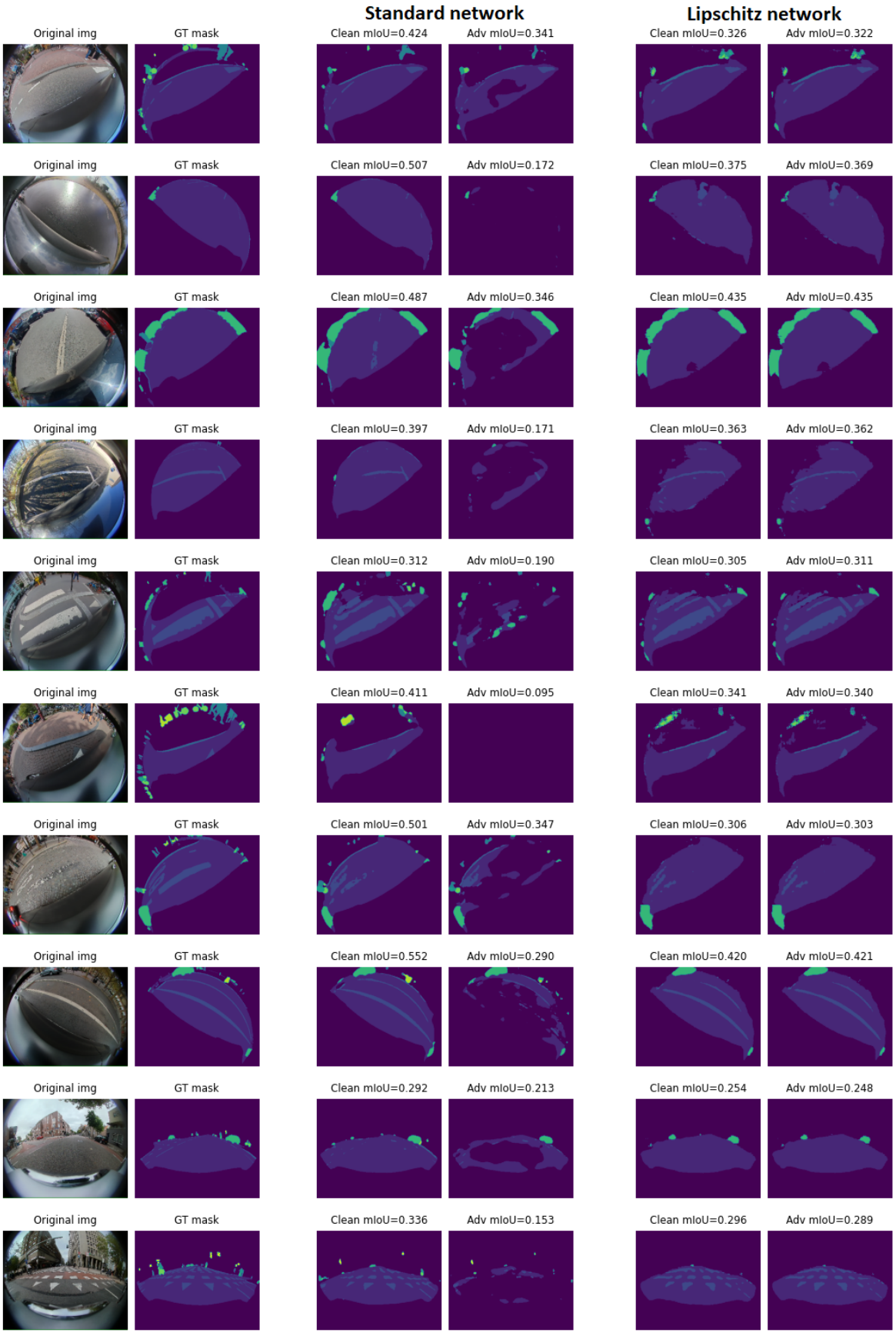


Figure G.8: Standard and robust segmentation on Valeo Woodscape. The two first columns show ten images from the validation set and their ground truth segmentation maps. The third and fourth columns are the prediction maps of the original and adversarial images for the standard network. The two last columns are the prediction maps for the robust 1-Lipschitz network.

Chapter H

Conclusion

In this document, we have shown Methods and Evaluation tools for Robust AI in Industrial applications. These theoretical and numerical findings are shared through this report in addition to the software and documentation that is delivered to the confiance.ai program. The guidelines and use case applications will thus be available in the confiance.ai environment.

The goal here was to share the tools and results that were obtained during the batch 2 of EC4. These methods often share the same use cases as a basis to demonstrate their performance. In this way, we see that all activities are strongly complementary.

For instance, the program will benefit from the progress of "Environment Alterations of AI models" with a particular focus on the **Renault welding Use case** under perturbations, such as patch attacks or ML watermarking. "Lipschitz networks" were also applied to the same use case. This latter project is an example of a methodological contribution that also provides an accessible open-source software aiming at providing the best possible trade-off between accuracy and robustness.

From the uncertainty quantification standpoint, EC4 has developed beyond state of the art techniques on both a method independent of the predictive model i.e. "Conformal prediction" and a tool specific for deep learning called "Bayesian Neural Networks". Both pieces of software are available for the confiance.ai program with application examples on the **Air Liquide demand forecast use case**.

We also provide guidelines on the use of "Robustification methods based on Neural Differential Equations" for the **Air Liquide Cylinder Counting**.

Chapter I

Bibliography

Bibliography

El Mehdi Achour, François Malgouyres, and Franck Mamalet. Existence, stability and scalability of orthogonal convolutional neural networks, 2021. URL https://arxiv.org/abs/2108.05623.

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In 27th USENIX Security Symposium (USENIX Security 18), 2018.

Ahmed M. Alaa and Mihaela van der Schaar. Frequentist uncertainty in recurrent neural networks via blockwise influence functions. CoRR, abs/2006.13707, 2020. URL https://arxiv.org/abs/2006.13707.

David J. Aldous. Exchangeability and related topics. In École d'Été de Probabilités de Saint-Flour XIII — 1983, pages 1–198. Springer, 1985.

Brian DO Anderson. Reverse-time diffusion equation models. Stochastic Processes and their Applications, 12(3):313–326, 1982.

Anastasios Nikolas Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. arXiv:2107.07511v3, 2021.

Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-dpm: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. ICLR, 2022.

Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLOS ONE, 12(7):1–24, 07 2017. doi: 10.1371/journal.pone.0180944. URL https://doi.org/10.1371/journal.pone.0180944.

Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. Predictive inference with the jackknife+. The Annals of Statistics, 49(1):486 – 507, 2021.

Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. Conformal prediction beyond exchangeability. arXiv preprint arXiv:2202.13415, 2022.

Ioannis K Bazionis and Pavlos S Georgilakis. Review of deterministic and probabilistic wind power forecasting: Models, methods, and future research. Electricity, 2(1):13–47, 2021.

Philipp Benz, Soomin Ham, Chaoning Zhang, Adil Karjauv, and In So Kweon. Adversarial robustness comparison of vision transformer and mlp-mixer to cnns. arXiv preprint arXiv:2110.02797, 2021.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In Joint European conference on machine learning and knowledge discovery in databases, pages 387–402. Springer, 2013.

Christopher M. Bishop. Mixture density networks. WorkingPaper, Aston University, 1994. Backup Publisher: Aston University ISBN: NCRG/94/004.

Henrik Boström, Henrik Linusson, Tuve Löfström, and Ulf Johansson. Accelerating difficulty estimation for conformal regression forests. Annals of Mathematics and Artificial Intelligence, 81(1–2):125–144, oct 2017.

Léon Bottou and Olivier Bousquet. The Tradeoffs of Large Scale Learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, Advances in Neural Information Processing Systems, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf.

Leo Breiman. Bagging predictors. Machine learning, 24(2):123–140, 1996a.

Leo Breiman. Out-of-bag estimation. Technical report, Technical report, Statistics Department, University of California Berkeley, 1996b.

Nicolas Brosse, Alain Durmus, and Eric Moulines. The promises and pitfalls of Stochastic Gradient Langevin Dynamics. In Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/335cd1b90bfa4ee70b39d08a4ae0cf2d-Abstract.html.

Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. arXiv preprint arXiv:1712.09665, 2017a.

Tom B. Brown, Dandelion Mané, Martín Abadi Aurko Roy, and Justin Gilmer. Adversarial patch. In Conference on Neural Information Processing Systems, 2017b. URL https://machine-learning-and-security.github.io/papers/mlsec17_paper_27.pdf.

Louis Béthune, Thibaut Boissin, Mathieu Serrurier, Franck Mamalet, Corentin Friedrich, and Alberto González-Sanz, 2021. URL https://arxiv.org/abs/2104.05097.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In European conference on computer vision, pages 213–229. Springer, 2020.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp), pages 39–57. IEEE, 2017.

Nicholas Carlini, Florian Tramer, J Zico Kolter, et al. (certified!!) adversarial robustness for free! arXiv preprint arXiv:2206.10550, 2022.

D. M. Ceperley and M. Dewing. The penalty method for random walks with uncertain energies. The Journal of Chemical Physics, 110(20):9812–9820, May 1999. ISSN 0021-9606, 1089-7690. doi: 10.1063/1.478034. URL http://aip.scitation.org/doi/10.1063/1.478034.

Nicolo Cesa-Bianchi and Gabor Lugosi. Prediction, Learning, and Games. Cambridge University Press, Cambridge, 2006. ISBN 9780521841085. doi: 10.1017/CBO9780511546921. URL https://www.cambridge.org/core/books/prediction-learning-and-games/A05C9F6ABC752FAB8954C885D0065C8F.

Françoise Chaitin-Chatelin and Valérie Frayssé. Lectures on finite precision computations. SIAM, 1996.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence, 40(4): 834–848, 2017.

Qi Chen, Lei Wang, Yifan Wu, Guangming Wu, Zhiling Guo, and Steven L Waslander. Aerial imagery for roof segmentation: A large-scale dataset towards automatic mapping of buildings. ISPRS Journal of Photogrammetry and Remote Sensing, 147:42–55, 2019a.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. In Neural Information Processing Systems (NeurIPS), 2019b. URL http://arxiv.org/abs/1806.07366.

Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic Gradient Hamiltonian Monte Carlo. In Eric P. Xing and Tony Jebara, editors, Proceedings of the 31st International Conference on Machine Learning, volume 32 of Proceedings of Machine Learning Research, pages 1683–1691, Bejing, China, June 2014. PMLR. URL https://proceedings.mlr.press/v32/cheni14.html. Issue: 2.

Victor Chernozhukov, Kaspar Wüthrich, and Zhu Yinchu. Exact and robust conformal inference methods for predictive machine learning with dependent data. In Proceedings of the 31st Conference On Learning Theory, 2018.

Victor Chernozhukov, Kaspar Wüthrich, and Yinchu Zhu. Distributional conformal prediction. Proceedings of the National Academy of Sciences, 118(48), 2021.

Haoyu Chu, Shikui Wei, Qiming Lu, and Yao Zhao. Improving Neural ODEs via Knowledge Distillation. arXiv preprint arXiv:2203.05103, 2022.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In International Conference on Machine Learning, pages 1310–1320. PMLR, 2019a.

Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. CoRR, abs/1902.02918, 2019b. URL http://arxiv.org/abs/1902.02918.

R. Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. Technometrics, 22(4):495–508, 1980. ISSN 00401706. URL http://www.jstor.org/stable/1268187.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. CoRR, abs/2003.01690, 2020. URL https://arxiv.org/abs/2003.01690.

Francesco Croce, Sven Gowal, Thomas Brunner, Evan Shelhamer, Matthias Hein, and Taylan Cemgil. Evaluating the Adversarial Robustness of Adaptive Test-time Defenses. arXiv preprint arXiv:2202.13711, 2022.

Florence de Grancey, Jean-Luc Adam, Lucian Alecu, Sébastien Gerchinovitz, Franck Mamalet, and David Vigouroux. Object detection with probabilistic guarantees: A conformal prediction approach. In Mario Trapp, Erwin Schoitsch, Jérémie Guiochet, and Friedemann Bitsch, editors, Computer Safety, Reliability, and Security. SAFECOMP 2022 Workshops, pages 316–329. Springer International Publishing, 2022. ISBN 978-3-031-14862-0.

Jacopo Diquigiovanni, Matteo Fontana, and Simone Vantini. Distribution-free prediction bands for multivariate functional time series: an application to the italian gas market, 2021.

Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Taylor-Lagrange Neural Ordinary Differential Equations: Toward Fast Training and Evaluation of Neural ODEs. arXiv preprint arXiv:2201.05715, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs, 2019.

Shai Feldman, Stephen Bates, and Yaniv Romano. Conformalized online learning. arXiv preprint arXiv:2205.09095v4, 2022. URL https://arxiv.org/abs/2205.09095v4.

Adeline Fermanian, Pierre Marion, Jean-Philippe Vert, and Gérard Biau. Framing RNN as a Kernel Method: A Neural ODE Approach. Advances in Neural Information Processing Systems, 34, 2021.

Rina Foygel Barber, Emmanuel J Candès, Aaditya Ramdas, and Ryan J Tibshirani. The limits of distribution-free conditional predictive inference. Information and Inference: A Journal of the IMA, 10(2):455–482, 2020.

Adrià Garriga-Alonso and Vincent Fortuin. Exact Langevin Dynamics with Stochastic Gradients. In Third Symposium on Advances in Approximate Bayesian Inference, 2021. URL https://openreview.net/forum?id=Rprd8aVUYkE.

Isaac Gibbs and Emmanuel Candès. Adaptive conformal inference under distribution shift. Advances in Neural Information Processing Systems, 34:1660–1672, 2021.

Isaac Gibbs and Emmanuel Candès. Conformal inference for online prediction with arbitrary distribution shifts. arXiv preprint arXiv:2208.08401, 2022.

Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin Cubuk. Adversarial Examples are a Natural Consequence of Test Error in Noise. In International Conference on Machine Learning, pages 2280–2289. PMLR, 2019.

Martin Gonzalez, Thibault Defourneau, Hatem Hajri, and Mihaly Petreczky. Realization Theory Of Recurrent Neural ODEs Using Polynomial System Embeddings. arXiv preprint arXiv:2205.11989, 2022a.

Martin Gonzalez, Hatem Hajri, Loic Cantat, and Mihaly Petreczky. Noisy learning for neural odes acts as a robustness locus widening. International Conference in Machine Learning (PODS Workshop), 2022b.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations, 2015. URL http://arxiv.org/abs/1412.6572.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.

Chirag Gupta, Arun K Kuchibhotla, and Aaditya K Ramdas. Nested conformal prediction and quantile out-of-bag ensemble methods. arXiv preprint arXiv:1910.10562, 2019.

Gerald J. Hahn and William Q. Meeker. Statistical intervals: a guide for practitioners. John Wiley & Sons, Inc., 1991.

Frank R. Hampel. The influence curve and its role in robust estimation. Journal of the American Statistical Association, 69(346):383–393, 1974. ISSN 01621459. URL http://www.jstor.org/stable/2285666.

T. Hastie, R. Tibshirani, and J.H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, second edition edition, 2009.

Elad Hazan et al. Introduction to online convex optimization. Foundations and Trends® in Optimization, 2(3-4):157–325, 2016.

Dan Hendrycks and Thomas Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. Proceedings of the International Conference on Learning Representations, 2019.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The Many Faces Of Robustness: A Critical Analysis Of Out-Of-Distribution Generalization. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 8340–8349, 2021.

Joeri Hermans, Arnaud Delaunoy, François Rozet, Antoine Wehenkel, and Gilles Louppe. Averting A Crisis In Simulation-Based Inference, October 2021. URL http://arxiv.org/abs/2110.06581. Number: arXiv:2110.06581 arXiv:2110.06581 [cs, stat].

Yifei Huang, Yaodong Yu, Hongyang Zhang, Yi Ma, and Yuan Yao. Adversarial Robustness of Stabilized Neural ODE Might be from Obfuscated Gradients. In Proceedings of the 2nd Mathematical and Scientific Machine Learning Conference, volume 145 of Proceedings of Machine Learning Research, pages 497–515. PMLR, 2022.

Ivan, Aaron, and Yisong Yue. LyaNet: A Lyapunov framework for training neural ODEs. arXiv pre-print server, 2022. doi: Nonearxiv:2202.02526.

Vilde Jensen, Filippo Maria Bianchi, and Stian Normann Anfinsen. Ensemble conformalized quantile regression for probabilistic time series forecasting. IEEE Transactions on Neural Networks and Learning Systems, pages 1–12, 2022. doi: 10.1109/TNNLS.2022.3217694.

Ulf Johansson, Henrik Boström, Tuve Löfström, and Henrik Linusson. Regression conformal prediction with random forests. Machine learning, 97(1-2):155–176, 2014.

Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. arXiv preprint arXiv:2105.14080, 2021.

Qiyu Kang, Yang Song, Qinxu Ding, and Wee Peng Tay. Stable Neural ODE with Lyapunov-Stable Equilibrium Points for Defending Against Adversarial Attacks. Advances in Neural Information Processing Systems, 34, 2021.

Katarzyna Kapusta, Vincent Thouvenot, and Olivier Bettan. Watermarking at the service of intellectualproperty rights of ml models. Conference on Artificial Intelligence for Defense (CAID), 2020.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. Nature Reviews Physics, 3(6):422–440, 2021.

P. Kidger. On Neural Differential Equations. PhD thesis, Oxford University, 2022.

Byol Kim, Chen Xu, and Rina Barber. Predictive inference is free with the jackknife+-after-bootstrap. In Advances in Neural Information Processing Systems, 2020.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. Advances in neural information processing systems, 34:21696–21707, 2021.

Roger Koenker and Gilbert Bassett Jr. Regression quantiles. Econometrica, 46(1):33–50, 1978.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions, 2017. URL https://arxiv.org/abs/1703.04730.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A Benchmark of in-the-Wild Distribution Shifts. In Marina Meila and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 5637–5664. PMLR, 18–24 Jul 2021.

Zhifeng Kong and Wei Ping. On fast sampling of diffusion probabilistic models. arXiv preprint arXiv:2106.00132, 2021.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops, June 2013.

Aditi S Krishnapriyan, Alejandro F Queiruga, N Benjamin Erichson, and Michael W Mahoney. Learning Continuous Models for Continuous Physics. arXiv preprint arXiv:2202.08494, 2022.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. 2017. URL https://arxiv.org/abs/1611.01236.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

Mathias Lécuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019, pages 656–672. IEEE, 2019. doi: 10.1109/SP.2019.00044. URL https://doi.org/10.1109/SP.2019.00044.

Mark Lee and Zico Kolter. On physical adversarial patches for object detection. arXiv preprint arXiv:1906.11897, 2019.

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. Journal of the American Statistical Association, 113(523):1094–1111, 2018. doi: 10.1080/01621459.2017.1307116. URL https://doi.org/10.1080/01621459.2017.1307116.

Pengcheng Li, Jinfeng Yi, Bowen Zhou, and Lijun Zhang. Improving the Robustness of Deep Neural Networks via Adversarial Training with Triplet Loss. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pages 2909–2915. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/403. URL https://doi.org/10.24963/ijcai.2019/403.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In International Conference on Learning Representations, 2021.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. Conformal prediction intervals with temporal dependence. Transactions of Machine Learning Research, 2022. URL https://openreview.net/forum?id=8QoxXTDcsH.

Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors. SafeAI 2019 (AAAI Workshop on Artificial Intelligence Safety), 2018.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

Giulio Lovisotto, Nicole Finnie, Mauricio Munoz, Chaithanya Kumar Mummadi, and Jan Hendrik Metzen. Give me your attention: Dot-product attention considered harmful for adversarial patch robustness. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 15234–15243, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps, 2022. URL https://arxiv.org/abs/2206.00927.

Eric Luhman and Troy Luhman. Knowledge distillation in iterative generative models for improved sampling speed. arXiv preprint arXiv:2101.02388, 2021.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018a. URL https://openreview.net/forum?id=rJzIBfZAb.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018b. URL https://openreview.net/forum?id=rJzIBfZAb.

Kaleel Mahmood, Rigel Mahmood, and Marten Van Dijk. On the robustness of vision transformers to adversarial examples. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 7838–7847, 2021.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting Neural ODEs. arXiv preprint arXiv:2002.08071, 2020.

Takashi Matsubara, Yuto Miyatake, and Takaharu Yaguchi. Symplectic Adjoint Method for Exact Gradient of Neural ODE with Minimal Memory. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021. URL https://openreview.net/forum?id=46J_l-cpc1W.

Nicolai Meinshausen. Quantile regression forests. Journal of Machine Learning Research, 7 (35):983–999, 2006.

Mouhcine Mendil, Luca Mossina, Marc Nabhan, and Kevin Pasini. Robust gas demand forecasting with conformal prediction. In Ulf Johansson, Henrik Boström, Khuong An Nguyen, Zhiyuan Luo, and Lars Carlsson, editors, Proceedings of the Eleventh Symposium on Conformal and Probabilistic Prediction with Applications, volume 179 of Proceedings of Machine Learning Research, pages 169–187. PMLR, 24–26 Aug 2022. URL https://proceedings.mlr.press/v179/mendil22a.html.

S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2574–2582, 2016. doi: 10.1109/CVPR.2016.282.

S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 86–94, July 2017. doi: 10.1109/CVPR.2017.17.

Norman Mu and Justin Gilmer. MNIST-C: A Robustness Benchmark for Computer Vision. In ICML Workshop on Uncertainty and Robustness in Deep Learning, 2019.

Marc Nabhan, Luca Mossina, Mouhcine Mendil, Kevin Pasini, Ansgar Radermarcher, Arnez Fabio, Olivier Galibert, and Daniel Régis Sarmento Caon. Project ec3, action sheet 5:predictive uncertaintyquantification and calibration. Technical report, Confiance.ai Program, 2022.

Federico Nesti, Giulio Rossolini, Saasha Nair, Alessandro Biondi, and Giorgio Buttazzo. Evaluating the robustness of semantic segmentation for autonomous driving against real-world adversarial patch attacks. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 2280–2289, 2022.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 427–436, 2015.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In International Conference on Machine Learning, pages 8162–8171. PMLR, 2021.

Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Anima Anandkumar. Diffusion models for adversarial purification. In International Conference on Machine Learning (ICML), 2022.

Katharina Ott, Prateek Katiyar, Philipp Hennig, and Michael Tiemann. ResNet After All: Neural ODEs and Their Numerical Solution. In International Conference on Learning Representations, 2021.

Ozan Özdenizci and Robert Legenstein. Restoring vision in adverse weather conditions with patch-based denoising diffusion models. arXiv preprint arXiv:2207.14626, 2022.

Avik Pal, Alan Edelman, and Christopher Rackauckas. Mixing Implicit and Explicit Deep Learning with Skip DEQs and Infinite Time Neural ODEs (Continuous DEQs). arXiv preprint arXiv:2201.12240, 2022.

Jingyue Pang, Datong Liu, Yu Peng, and Xiyuan Peng. Optimize the coverage probability of prediction interval for anomaly detection of sensor-based monitoring series. Sensors, 18(4): 967, 2018.

Harris Papadopoulos and Haris Haralambous. Reliable prediction intervals with regression neural networks. Neural Networks, 24(8):842–851, 2011.

Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive confidence machines for regression. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, Machine Learning: ECML 2002, pages 345–356, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-36755-0.

Harris Papadopoulos, Vladimir Vovk, and Alexander Gammerman. Regression conformal prediction with nearest neighbours. Journal of Artificial Intelligence Research, 40:815–840, 2011.

N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In 2016 IEEE European Symposium on Security and Privacy (EuroS P), pages 372–387, March 2016. doi: 10.1109/EuroSP.2016.36.

Agustin Martin Picard, David Vigouroux, Petr Zamolodtchikov, Quentin Vincenot, Jean-Michel Loubes, and Edouard Pauwels. Leveraging Influence Functions for Dataset Exploration and Cleaning. In 11th European Congress Embedded Real Time Systems (ERTS 2022), pages 1–8, Toulouse, France, June 2022. URL https://hal.archives-ouvertes.fr/hal-03617649.

Alejandro Francisco Queiruga, N. Benjamin Erichson, Liam Hodgkinson, and Michael W. Mahoney. Stateful ODE-Nets using Basis Function Expansions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021.

Dongwei Ren, Wangmeng Zuo, Qinghua Hu, Pengfei Zhu, and Deyu Meng. Progressive image deraining networks: A better and simpler baseline. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3937–3946, 2019.

Yaniv Romano, Evan Patterson, and Emmanuel Candès. Conformalized quantile regression. In Advances in Neural Information Processing Systems, 2019.

Yaniv Romano, Rina Foygel Barber, Chiara Sabatti, and Emmanuel Candès. With Malice Toward None: Assessing Uncertainty via Equalized Coverage. Harvard Data Science Review, 2 (2), apr 30 2020. https://hdsr.mitpress.mit.edu/pub/qedrwcz3.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10684–10695, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer, 2015.

Aniruddha Saha, Akshayvarun Subramanya, Koninika Patil, and Hamed Pirsiavash. Role of spatial context in adversarial robustness for object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pages 784–785, 2020.

Mohammadreza Salehi, Hossein Mirzaei, Dan Hendrycks, Yixuan Li, Mohammad Hossein Rohban, and Mohammad Sabokrou. A Unified Survey on Anomaly, Novelty, Open-Set, and Out-of-Distribution Detection: Solutions and Future Challenges. arXiv preprint arXiv:2110.14051, 2021.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. arXiv preprint arXiv:2202.00512, 2022.

C. Saunders, A. Gammerman, and V. Vovk. Transduction with confidence and credibility. In Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'99, page 722–726, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

Mathieu Serrurier, Franck Mamalet, Thomas Fel, Louis Béthune, and Thibaut Boissin. When adversarial attacks become interpretable counterfactual explanations, 2022. URL https://arxiv.org/abs/2206.06854.

Glenn Shafer and Vladimir Vovk. A Tutorial on Conformal Prediction. Journal of Machine Learning Research, 9(3), 2008.

Rulin Shao, Zhouxing Shi, Jinfeng Yi, Pin-Yu Chen, and Cho-Jui Hsieh. On the adversarial robustness of vision transformers. arXiv preprint arXiv:2103.15670, 2021.

Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l2 robustness on cifar-10 and cifar-100. In International Conference on Learning Representations, 2021.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. arXiv:2010.02502, October 2020. URL https://arxiv.org/abs/2010.02502.

Daniel A. De Souza, Diego Mesquita, Samuel Kaski, and Luigi Acerbi. Parallel MCMC Without Embarrassing Failures. In Proceedings of The 25th International Conference on Artificial Intelligence and Statistics, pages 1786–1804. PMLR, May 2022. URL https://proceedings.mlr.press/v151/de-souza22a.html. ISSN: 2640-3498.

Kamile Stankeviciute, Ahmed M Alaa, and Mihaela van der Schaar. Conformal time-series forecasting. Advances in Neural Information Processing Systems, 34:6216–6228, 2021.

Amos Storkey. When training and test sets are different: characterizing learning transfer. Dataset shift in machine learning, 30:3–28, 2009.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-Time Training with Self-Supervision for Generalization under Distribution Shifts. In Hal Daumé III and Aarti Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 9229–9248. PMLR, 13–18 Jul 2020.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

Hideyuki Tachibana, Mocho Go, Muneyoshi Inahara, Yotaro Katayama, and Yotaro Watanabe. Quasi-taylor samplers for diffusion generative models based on ideal derivatives, 2021. URL https://arxiv.org/abs/2112.13339.

Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring Robustness to Natural Distribution Shifts in Image Classification. Advances in Neural Information Processing Systems, 33:18583–18599, 2020.

Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pages 0–0, 2019.

Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candès, and Aaditya Ramdas. Conformal prediction under covariate shift. Advances in neural information processing systems, 32, 2019.

F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. USENIX Security Symposium., 2016.

Asher Trockman and J Zico Kolter. Patches are all you need? arXiv preprint arXiv:2201.09792, 2022.

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In Advances in Neural Information Processing Systems, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/file/485843481a7edacbfce101ecb1e4d2a8-Paper.pdf.

Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, 2017.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. Algorithmic learning in a random world. Springer Science & Business Media, 2005.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully Test-Time Adaptation by Entropy Minimization. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

Jiayun Wang, Yubei Chen, Rudrasis Chakraborty, and Stella X Yu. Orthogonal convolutional neural networks. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR'20, 2020.

Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. CoRR, abs/1807.01473, 2018. URL http://arxiv.org/abs/1807.01473.

Tianhao Wang and Florian Kerschbaum. Attacks on digital watermarks for deep neural networks. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

Larry Wasserman. All of Statistics. Springer, 2004.

Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. Learning fast samplers for diffusion models by differentiating through sample quality. In International Conference on Learning Representations, 2021.

Max Welling and Yee Whye Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. page 8.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A Multi-Horizon Quantile Recurrent Forecaster. arXiv e-prints, art. arXiv:1711.11053, November 2017.

Andrew G Wilson and Pavel Izmailov. Bayesian Deep Learning and a Probabilistic Perspective of Generalization. In Advances in Neural Information Processing Systems, volume 33, pages 4697–4708. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/322f62469c5e3c7dc3e58f5a4d1ea399-Abstract.html.

Olivier Wintenberger. Optimal learning with bernstein online aggregation. Machine Learning, 106(1):119–141, 2017.

Wojciech Wisniewski, David Lindsay, and Sian Lindsay. Application of conformal prediction interval estimations to market makers' net positions. In Proceedings of the Ninth Symposium on Conformal and Probabilistic Prediction and Applications, 2020.

Chen Xu and Yao Xie. Conformal prediction interval for dynamic time-series. In Marina Meila and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 11559–11569. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/xu21h.html.

Chen Xu and Yao Xie. Conformal prediction for time series. arXiv preprint arXiv:2010.09107v13, 2022.

Winnie Xu, Ricky TQ Chen, Xuechen Li, and David Duvenaud. Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations. In International Conference on Artificial Intelligence and Statistics, pages 721–738. PMLR, 2022.

Hanshu Yan, Jiawei Du, Vincent Tan, and Jiashi Feng. On Robustness of Neural Ordinary Differential Equations. In International Conference on Learning Representations, 2020.

Guoguo Yang, Kevin Burrage, Yoshio Komori, and Xiaohua Ding. A new class of structure-preserving stochastic exponential runge-kutta integrators for stochastic differential equations. BIT Numerical Mathematics, pages 1–33, 2022.

Margaux Zaffran, Olivier Feron, Yannig Goude, Julie Josse, and Aymeric Dieuleveut. Adaptive conformal predictions for time series. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pages 25834–25866. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/zaffran22a.html.

Gianluca Zeni, Matteo Fontana, and Simone Vantini. Conformal prediction: a unified review of theory and new challenges. arXiv preprint arXiv:2005.07972, 2020.

Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc P. Stoecklin, Heqinq Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 2018.

Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. arXiv preprint arXiv:2204.13902, 2022.

Qinsheng Zhang, Molei Tao, and Yongxin Chen. gddim: Generalized denoising diffusion implicit models. arXiv preprint arXiv:2206.05564, 2022.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2881–2890, 2017.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In International Conference on Learning Representations, 2020. URL https://openreview.net/forum?id=ryxmb1rKDS.