Leveraging Influence Functions for Dataset Exploration and Cleaning

Agustin Martin Picard, David Vigouroux, Petr Zamolodtchikov, Quentin Vincenot, Jean-Michel Loubes, Edouard Pauwels

Leveraging Influence Functions for Dataset Exploration and Cleaning

Agustin Martin Picard*†, David Vigouroux†, Petr Zamolodtchikov‡,
Quentin Vincenot‖†, Jean-Michel Loubes§, Edouard Pauwels¶
* Scalian, † IRT Saint Exupéry, ‡ University of Twente, ‖ Thales Alenia Space,
§ Institut Mathématique de Toulouse, Université Paul Sabatier, ¶ IRIT, CNRS, Université de Toulouse,

***Abstract*—In this paper, we tackle the problem of finding potentially problematic samples and complex regions of the input space for large pools of data without any supervision, with the objective of being relayed to and validated by a domain expert. This information can be critical, as even a low level of noise in the dataset may severely bias the model through spurious correlations between unrelated samples, and under-represented groups of data-points will exacerbate this issue. As such, we present two practical applications of influence functions in neural network models to industrial use-cases: exploration and clean-up of mislabeled examples in datasets. This robust statistics tool allows us to approximately know how different an estimator might be if we slightly changed the training dataset. In particular, we apply this technique to an ACAS Xu neural network surrogate model use-case[14] for complex region exploration, and to the CIFAR-10 canonical RGB image classification problem[20] for mislabeled sample detection with promising results.**

Keywords: Intelligent Systems, Artificial Neural Networks, Influence functions, Dataset Exploration, Mislabeled Sample Detection.

I. Introduction

In recent times, the field of artificial intelligence (AI) and machine learning has been shifting to a paradigm with an extreme reliance on huge, over-parametrized models and equally large pools of data. These models leverage correlations between samples of the same class to perform their predictions with an impressive precision on domains ranging from image classification [20, 8], object detection [8, 22] and semantic segmentation [22, 6] in computer vision, to sentiment analysis [23] and natural language translation [16] in natural language processing (NLP). The exponential growth of widely available, vast amounts of data for these applications has been a major catalyst in this process, and along with it comes the need to validate it and clean it.

Traditionally, domain experts would be requested to perform this task, but when dealing with millions of examples, it becomes impossible to realistically verify each one. As such, there has been an emergence of different techniques to remedy this issue by automatically proposing interesting samples that might need to be verified before integration into the dataset.

Related work

Namely, Koh et al. recovered a classical statistics tool and adapted it to differentiable statistical models and applied it to dataset cleaning and white-box model explainability [18] with interesting results in the former task. It is this formulation that we will be employing in this work. They later went further and studied the effect groups of data-points have on models in [17], with Basu et al. providing a second-order formulation that takes into account pairwise interactions between samples [3]. Choosing a different starting point, Giordano et al. have proposed alternative formulations based on the infinitesimal jack-knife for first [9] and higher order approximations [10], where they also provide bounds on the error incurred during the approximation.

More recently, Kong et al. have adapted the notion of influence function to Variational AutoEncoders in [19], where they address the problem of computing the loss of test samples over the expectation over the encoder, and they apply it to dataset cleaning tasks on the MNIST [21] and CIFAR-10 [20] computer vision datasets. With another application in mind, Alaa et al. have employed higher-order influence functions to measure uncertainty in deep learning models [1], showcasing the numerous applications of this tool.

There are also numerous techniques for estimating the influence of training samples based on other principles. In [32], the authors propose to leverage the representer theorem [26] to find the training points that contribute the most – both positive and negatively – to the model's output. In particular, they apply this technique to offer explanations to the neural network's decisions. With the same application in mind and basing themselves off of the same principle, Sui et al. derive a technique that employs a local Jacobian Taylor expansion instead of re-fitting a new output layer [30] – as in [32]. They also measure its capacity to detect mislabeled examples in very noisy scenarios. Finally, in [25], Pruthi et al. propose to compute a first-order approximation of the influence function by capitalizing on the model's checkpoints saved during the training process. This allows them to spare themselves the difficulties related to handling with hessian matrices of considerable size – a problem that, depending on the dimensionality of the task, can complicate matters in our case.

However, they all performed tests on canonical datasets and focused on single model architectures. In contrast, in this work, we successfully demonstrate the utility of influence functions on practical industrial applications, namely on the ACAS Xu use-case, and we perform experiments on multiple types of models and training schedules to conclude that they constitute a critical component to

obtaining satisfactory results.

In practice, we know that mislabeled data-points heavily influence the shape of the decision boundary, and thus, when fixed, force the optimal model to change drastically to accommodate this deformation. In much the same way, removing samples from insufficiently dense regions of the input space will result in a model whose parameters are significantly different. However, it would be prohibitively time-consuming to re-train a model leaving out one sample at a time in most use-cases. This is why we turned to the concept of the influence function, that allows us to approximately measure the importance of each data-point without the need for expensive re-trainings – assuming certain criteria are met.

II. Influence Functions

A. Theory

In the literature of classical robust estimation, the question of how to appropriately measure the influence of subsamples of data on a given estimator is often raised [11, 5, 13]. In other words, how an estimator would behave if some of the data-points were not provided. An accurate way to answer this question would be to re-fit the estimator on a dataset with these points held-out. However, for modern models on high-dimensional data, this process can be computationally very expensive and time-consuming. Fortunately, when working with neural network models, it is possible to leverage any of the modern, widely available *automatic differentiation* frameworks to approximate the answer.

In [18, 17], the authors adapt the concept of influence function to neural networks trained using empirical risk minimization (ERM) and leveraging the automatic calculation of gradients.

Definition 1. *Let $z_1, ..., z_n$ be a group of training points, where $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ – where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output spaces respectively –, $\theta \in \Theta$ be a set of parameters, $\ell(z, \theta)$ a loss function, and $\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(z_i, \theta)$ the empirical risk minimizer. Then, they posit that the influence that a test point z_{test} has in the model's weights induced by infinitesimally up-weighting a data-point z is equal to [18]:*

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = \nabla_\theta \ell\left(z_{\text{test}}, \hat{\theta}\right)^T H_{\hat{\theta}}^{-1} \nabla_\theta \ell\left(z, \hat{\theta}\right), \quad (1)$$

where $H_{\hat{\theta}} = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 \ell\left(z_i, \hat{\theta}\right)$ is the average Hessian with respect to the model's weights.

This formulation requires us to make some very strong assumptions about the underlying problem. Namely, we assume:

1) the loss $\ell : (x, y, \theta) \mapsto \ell(f_\theta(x), y)$ to be convex and twice continuously differentiable.
2) that $\exists \theta_0 \in \mathbb{R}^d$ such that $\nabla_\theta \mathbb{E}\left[\ell\left(f_{\theta_0}(x), y\right)\right] = 0$.
3) the matrix $H_{\hat{\theta}}$ to be invertible.

These hypotheses are reasonable when the model has been trained long enough to be close to a local first and second-order minimum. However, they can constrain the applicability of this formulation considerably, but to work around them, we may limit ourselves to the regression – linear or logistic, depending on the task – between the embedding engendered by the feature extraction chain (from the input to the last layer of the neural network) and the output. If this embedding was created carefully enough, the first and last assumptions should be true, and the second one, approximately verified ($\nabla_\theta \mathbb{E}\left[\ell\left(f_{\theta_0}(x), y\right)\right] \approx 0$). This also allows us to drastically reduce the computational cost associated to these calculations for the large models that are typically used in high-dimensional applications.

Returning to Eq. 1, when $z = z_{\text{test}}$, this quantity is exactly identical to the influence measure introduced by Cook in [5], also known as Cook's distance, and we will apply this concept to our practical applications.

B. Implementation

Up until now, we have considered a theoretical problem leaving out all possible practical issues we might run into, but it turns out that the exact computation of Cook's distance through the formulation we described above can raise plenty of technical challenges. Namely, calculating the hessian – and thus, its inverse – can be extremely computationally intensive, and even impossible depending on the amount of weights our model has, as this matrix grows quadratically in size with the model's parameters.

The first measure to alleviate the computational burden is to reduce the amount of parameters to consider during the estimation of the $H_{\hat{\theta}}^{-1}$. In particular, we considered only the neural network's last layer containing trainable weights, thus both reducing the size of the hessian and verifying the hypothesis we need for accurate calculation of these quantities.

Secondly, when this does not suffice to render the problem tractable, it is still possible to estimate the *inverse-hessian-vector product* directly without explicitly computing the hessian by using a Conjugate Gradient Descent algorithm [29]. Furthermore, by leveraging *forward-over-backward automatic differentiation*, we can do so in a very memory-efficient manner. However, although this scheme has been successfully implemented, it was not employed for the results we showcase below; **we do estimate the hessian matrix of the loss with respect to the model's weights in our experiments**.

C. Methodology

Intuitively, we can use the concept of influence introduced in 1 to obtain a measure of how "interesting" a data-point in the training dataset is to a given model. If we consider that those that are difficult to learn will be so for any statistical model, we can use the information relayed by these influence functions to assign a value of interest to each data-point and provide them to the user for validation.

It is important to note that we will be focusing on the influence functions with respect to the neural network's last layer. This not only renders the computation considerably more memory-efficient, but also places us closer to the hypotheses that are necessary for accurate computation of these values.

In all of our experiments and for both of the different applications – detection of important data-points and mislabeled example detection –, the procedure we followed was to:

1) Train a neural network model until convergence on 80%/20% training/test dataset splits as it is conventionally done.
2) Compute the value of Cook's distance for each training point separately.
3) Sort the training dataset by its data-points' Cook's distance.
4) Consider the top-most points – i.e. those whose Cook's distance was the highest – as important or mislabeled, and **request human input for validation**.

By performing this procedure, we intend to determine which training points are maximally important to the model, and without whom the final neural network would be the most different. This typically means samples that are under-represented in the dataset – as the model would rely heavily on the little information it has to learn how to predict in those cases –, mislabeled data-points – for much the same reason –, or those that lie right in the decision boundary – as they help the model fix it correctly. In our case, examples coming from all three situations are useful for our tasks.

Concerning the detection of mislabeled samples, as this technique aims to provide us with the training points that would change our model the most if they were removed, we posit that it could help in their detection. These data-points, that, due to a variety of conditions during the consolidation of the dataset, have labels that do not correspond to the correct classes, would introduce confounding factors to our model during the training process. As such, we want to rid ourselves of them, and to do so as efficiently as possible, as inspecting the whole dataset one example at a time might be prohibitively time-consuming for large pools of data. Thus, we postulate that by sorting our data-points by their Cook's distance value, we should be able to accelerate this process and find the mislabeled examples among the most influential points in the dataset.

As a matter of fact, this technique has already been used as a baseline for other methods [32, 25], but with what seems to be different model configurations and training schedules. Indeed, by testing out different architectures, *learning rate* decay functions and layer regularization strategies, we have found that these have a considerable impact on its capacity to single out mislabeled examples.

III. Results

For each of the tasks, we started by testing our intuitions on synthetically generated toy datasets, and then moved on to the actual use-cases. In particular, for the detection of interesting data-points, we focused on simple, two-dimensional binary classification datasets with an increasing level of complexity, and then applied what we had learned to a drone collision avoidance problem. For the mislabeled point detection, we followed the same procedure: we corroborated that we were able to correctly identify a single mislabeled data-point on a simple, two-dimensional binary classification synthetic dataset, and then moved on to noisy versions of the CIFAR-10 image classification dataset.

A. Detection of interesting training points

Experiments on toy examples

As a means to test the validity of this mathematical tool in the context of deep learning models, we began with some examples where we controlled perfectly the inputs and already knew what to expect – i.e. some toy examples. In particular, we generated groups of points in two-dimensional space, with different decision boundaries and local down-sampling strategies to corroborate our intuitions.

For all the following toy problems, a simple 2-hidden layer multi-layer perceptron with sigmoid activations was employed. They were all trained until convergence on splits with 80% of the whole data being used during the training process, and the remaining 20% was held out for validation. Then, we applied our methodology and plotted the whole training dataset with the transparency set as a function of each point's influence.

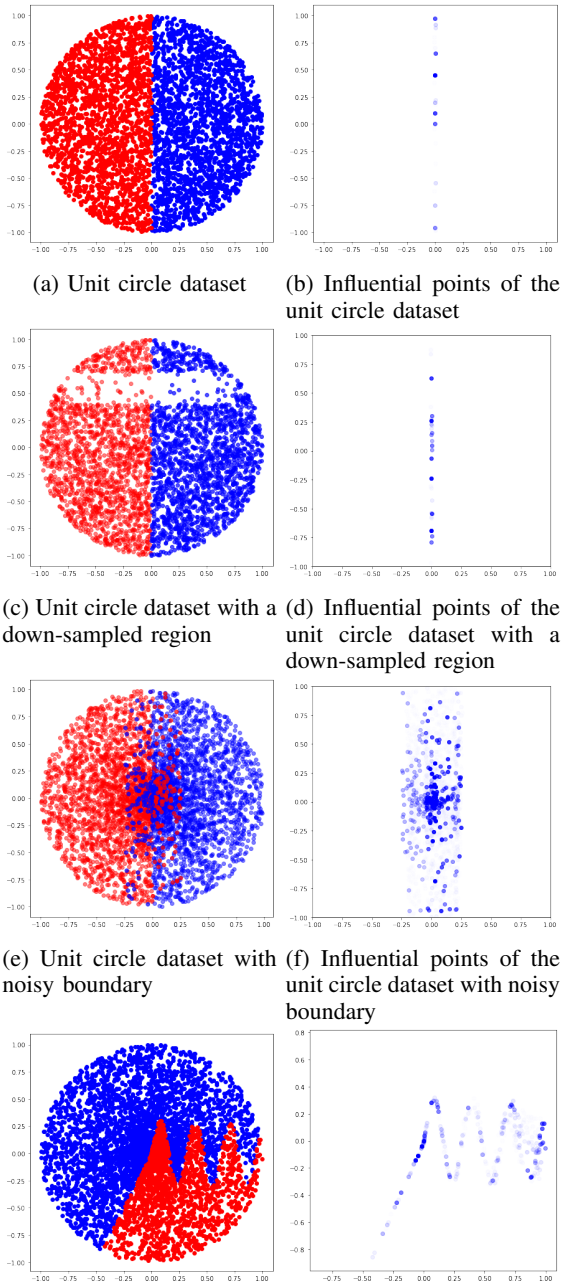
In Fig. 1, we showcase the results, and we observe that for each problem, the most influential points are always close to the decision boundary. This applies to the more complex examples as well, telling us that our intuitions about the capabilities of the power of this tool might be correct. In particular, it is interesting to note how the data-points near under-sampled parts of the boundary of the more complex example carry some influence, as otherwise the model would not know where to place its decision boundary. Similarly, we observe the whole region with mislabeled samples in the noisy boundary toy example to be influential, as the model is not quite confident inside of it and is forced to memorize each data-point, thus rendering them important to the model in question. This last intuition will be important for the mislabeling sample detection task later on.

Experiments on ACAS Xu

The ACAS Xu problem [14] is an unmanned drone collision avoidance use-case, where, knowing some information about the drone and the intruder's states, and the previous action that was taken, a **look-up table** (**LUT**) is employed to compute the next optimal step our drone has to take to successfully avoid a collision between the two aircraft. Considering that these decisions will depend on 7 variables – 6 describing the current state of both drones + 1 for the previous action –, this cost table can be quite difficult to manipulate efficiently, specially due to the fact that it contains 93 million points.

The six variables that are presented in Fig. 2 are:

- ρ [ft]: The distance from ownship to intruder.
- θ [rad]: The angle to intruder relative to ownship heading direction.
- ψ [rad]: The heading angle of the intruder relative to ownship heading direction.
- v_{own} [ft/s]: The speed of the ownship.
- v_{int} [t/s]: The speed of the intruder.
- τ [sec]: Time until the loss of vertical separation.



(a) Unit circle dataset

(b) Influential points of the unit circle dataset

(c) Unit circle dataset with a down-sampled region

(d) Influential points of the unit circle dataset with a down-sampled region

(e) Unit circle dataset with noisy boundary

(f) Influential points of the unit circle dataset with noisy boundary

(g) More complex 2D dataset with a line boundary that transforms into a sinus

(h) Influential points of the more complex 2D dataset.

Fig. 1: The synthetic 2D toy datasets and their corresponding most influential samples according to simple two-layer perceptrons, in increasing complexity from top to bottom. In the scatter plots on the left, each color indicates the class label for each point. On the right, all the points are colored blue and their transparency is a function of their influence value.

Consequently, a solution to reduce its cost is to train a neural network to operate as a surrogate model, which, in inference, would be able to produce predictions much faster and at a much lower memory cost. This technique would allow us to accurately approximate the predictions of the 2GB+ table with a simple ≈3MB neural network[14, 7].

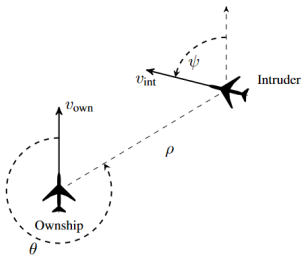


Fig. 2: Illustration of the ACAS Xu problem, as per [14].

Placing ourselves in this frame of work, we wish to increase the confidence we have on what the model is learning without having to manually search for regions of the input space that may be hard to learn for the model. As such, will apply the methodology described above to determine whether we need to gather more data on specific regions with the help of a human domain expert.

To do so, we trained a simple, 6-hidden-layer multi-layer perceptron with ReLU activations with an Adam optimizer on 8096 sized mini-batches with the table's contents for 200 epochs until convergence, reaching 99% accuracy. We chose to optimize directly for the classification between the different 5 actions – 'COC' (clear of conflict), 'WR' (weak right), 'WL' (weak left), 'R' (right) and 'L' (left) – and starting always from the state 'COC', to keep the optimization and interpretation of the results simple, as well as be able to easily apply our current formulation of the influence functions. Additionally, as it is typically done in these sorts of scenarios, we held out 20% of the dataset for validation, which left us with 80% for training.

Thus, we applied the influence function's method to single out the most influential datapoints in the training dataset. For these points, we plotted the two-dimensional cuts obtained by sweeping the range (i.e. the distance to the intruder) and the theta (i.e. the angle between the ownship and the intruder) while keeping the rest of the parameters constant. These plots simulate a situation where the intruder moves in a straight line and our drone tries to avoid it. We expect to find some interesting scenarios, and the most influential points to be close to the decision boundary.

We observe from Fig. 3 that these influential regions usually contain either groups of points that are quite close to each other but belong to different classes, or are sparsely sampled. In both cases, the influence is mostly monopolized by the samples that are close to the decision boundary.

Once these regions have been identified, we can present them to a domain expert for further analysis and validation. If they are indeed important for the task, more data could be gathered to facilitate the learning process near them, and if not, then they can be filtered out to prevent the addition of confounding data-points.

In particular, when we showed our results to some experts on the ACAS Xu drone collision avoidance problem, they

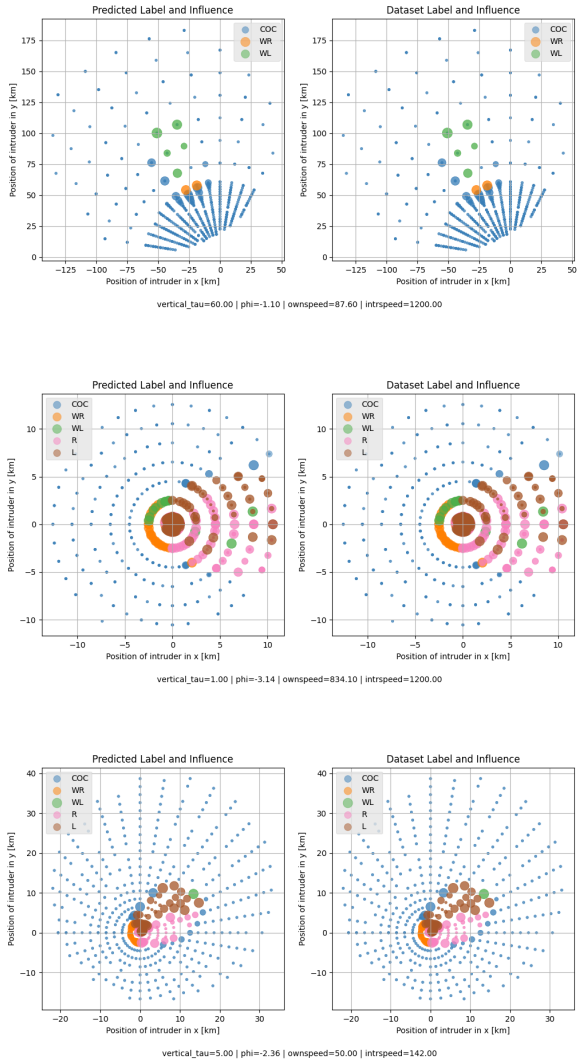


Fig. 3: Some 2D cuts of influential regions in the input space. The scatter plot on the right of each sub-figure represents the ground-truth's decisions, and the one on the left, the model's. Each dot is a point in the training dataset, and its size depends on its influence, bigger meaning more influential.

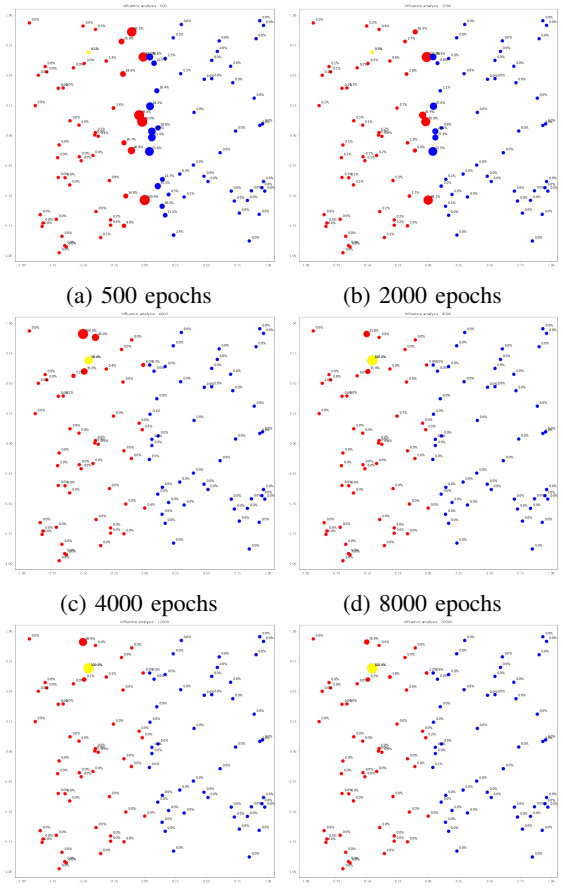
found some interesting patterns: in most of the most influential 2D cuts we generated, there was either one or both drones that were traveling at high speeds. Additionally, they found that there were some situations where the region was very sparsely sampled, and contained some very sudden changes in the **LUT**'s decisions – i.e. switching from left to right and back in contiguous data-points. This is important information to have before finishing the consolidation of the training dataset, as it could guide experts to push for asking for more data from the regions in input space in question.

B. Mislabeled sample detection

Experiment on a toy example

As we have done for the previous task, we test our intuitions on a synthetic dataset so as to be able to easily visualize and understand the problem at hand. In this case, we will generate a set of uniformly distributed points, with a decision boundary at the center of the horizontal axis, of which one point will have its label flipped.

In this experiment, we will attempt to trace the point's influence during the model's training phase to verify whether we can retrieve it successfully. Given the previous ones, we would expect the point to dominate the rest as the model starts to gradually *overfit* on it to minimize the training loss. As the problem is quite a simple one, we solve it with a simple, one-hidden-layer MLP, and we minimize a binary cross-entropy loss.



(a) 500 epochs (b) 2000 epochs

(c) 4000 epochs (d) 8000 epochs

(e) 12000 epochs (f) 20000 epochs

Fig. 4: Influence of a simple 2D binary classification dataset as training progresses. Blue and red indicate each class and the yellow data-point has been incorrectly labeled as blue in the red zone (i.e. has been mislabeled). The size of each point is proportional to its Cook's distance.

In Fig. 4, we observe that the model attributes a high influence to the decision boundary points at first, but as training progresses and it starts to overfit, the influence shifts

to the points close to the mislabeled one – showing that the model is forced to bend its decision boundary to accommodate this outlier –, and ends up with only the mislabeled point dominating the rest of the dataset in terms of influence value.

Experiments on CIFAR-10

Now that our intuitions have been confirmed, we will gauge the informative power of this technique on the CIFAR-10[20] RGB image classification dataset. This problem consists on assigning the correct category to natural images of 32×32 pixels and belonging to 10 classes ranging from cars to frogs and in different contexts.

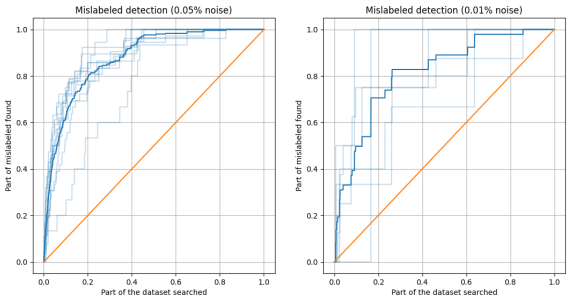
In particular, we have tested the **EfficientNetB0**[31], **ResNet-20**[12] and **VGG-19**[27] architectures, trained with Adam[15] optimizers and custom learning rate schedules, and Dropout[28] and layer-wise elastic net reguralizations (`L1L2`). For each of these configurations, we have trained sets of 6–12 models on the CIFAR-10[20] dataset on two noisy regimes: $\approx 0.05\%$ and $\approx 0.01\%$ of randomly changed labels throughout the training set. These proportions of noise were chosen to simulate what we would expect to have on standard, clean data in industrial use-cases.

In Fig. 5, we observe that, in most cases, our technique of searching the most influential samples for mislabeled images is a good strategy for cleaning up a slightly noisy dataset. Furthermore, it seems that the choice of architecture can have a considerable impact on the results.

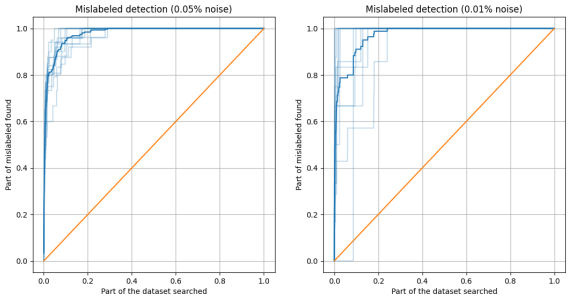
In particular, as we are comparing ROCs, we expect the best curves to be those that get as close to as possible to detecting every mislabeled image with as little samples as possible. Thus, we notice that the **EfficientNetB0** architecture does not seem suitable for this task as the **VGG-19**, which itself performs quite well if we exclude the outliers. Ideally, we would not have any poorly performing models from a given architecture, and this seems to be the case for the **ResNet-20** models. This is why we used this last architecture for the rest of the experiments.

Additionally, we tested the effect of adding elastic net regularization to the classification head of the ResNet-20, and of training with a smaller *learning rate* for more epochs. These results are presented in Fig. 6.

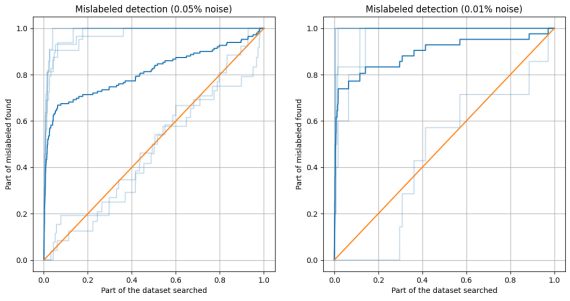
In [2], it was demonstrated that it is possible to more accurately compute influence values on layers that have been trained with layer-wise regularization, and in Fig. 6a, we corroborate this and we obtain an even better performance. However, contrary to our intuition, by training for a longer period of time – and potentially overfitting the model on the training set –, we severely deteriorate the network's capacity to retrieve these samples. We surmise that once it has reached convergence, the cross-entropy loss encourages the model to assemble all the points from each class together and form tightly knit clusters, and to separate these groups from each other as much as possible. This leads to it not being able to differentiate individual points through their influence, and hence, to not be capable of detecting these artificially generated mislabeled images.



(a) EfficientNetB0



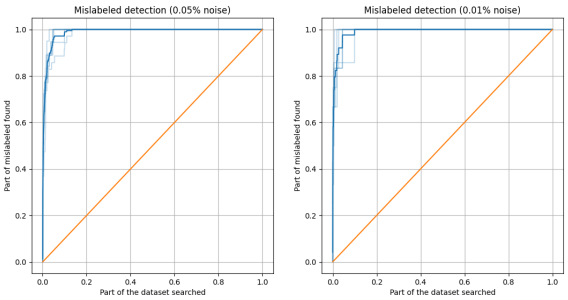
(b) ResNet-20



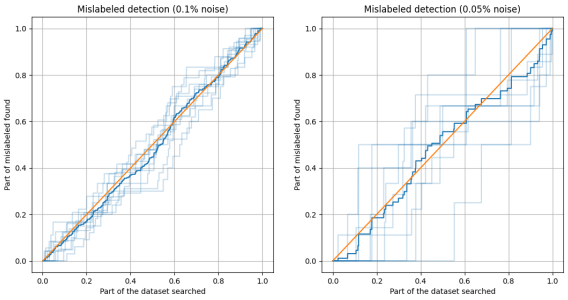
(c) VGG-19

Fig. 5: ROC curves for the detection of mislabeled examples for both noise regimes. In each case, we plot the mean ROC in solid blue, each individual ROC in transparent blue, and the random baseline in orange.

These two phenomena of performing better when constrained by the `L1L2` regularization and worse when overfitting are the two sides of the same coin: once an unconstrained network starts approaching convergence, it can decrease the surrogate classification loss (i.e. the negative log-likelihood or cross-entropy loss) without altering the 0-1 loss – the actual loss we would like to optimize, that indicates whether an element was correctly classified – by increasing its Lipschitz constant [4]. This can be easily demonstrated by leveraging some of the *softmax* activation function's properties when the neural network has already achieved the 100% accuracy in the training dataset, but this phenomenon has been observed to also occur slightly before reaching this point [4]. In layman terms, this means that the model will attempt to maximally



(a) ResNet-20 with `L1L2`



(b) ResNet-20 with slower schedule

Fig. 6: ROC curves for the detection of mislabeled examples for both noise regimes when training the model with an `L1L2` constraint, and a slower *learning rate* schedule. In each case, we plot the mean ROC in solid blue, each individual ROC in transparent blue, and the random baseline in orange.

distance the points from different classes, theoretically converging to N_{class} clusters of points infinitely far away from each other in the form of a simplex [24].

We leave for future work the analysis of when this happens and how to guarantee that our model has been correctly trained for useful information retrieval through influence functions.

IV. Conclusions

Initially applied to simple models and computed in its exact form, the influence function fell into disuse when models and datasets started to grow, making the procedure practically intractable. However, with the advent of frameworks that efficiently implement *auto-differentiation* and the popularization of GPUs capable of greatly accelerating computation times, it became possible to develop an approximate version specific to neural network models.

In this work, we have successfully leveraged them to determine influential points in the dataset and retrieve potentially problematic regions on the ACAS Xu use-case, and for the mislabeled example detection on the CIFAR-10 image classification dataset. In both cases, despite the size and complexity of the models at hand, the results were quite impressive, and come to show of its usefulness in real-life scenarios.

Acknowledgments

This work was conducted as part of the DEEL project[1]. Funding was provided by ANR-3IA Artificial and Natural Intelligence Toulouse Institute (ANR-19-PI3A-0004). The authors thank Florence de Grancey, Claire Pagetti and Adrien Gauffriau for their input on our results on the ACAS Xu problem.

References

[1] Mihaela van der Schaar Ahmed M. Alaa. Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions. In *International Conference on Machine Learning*, 2020.

[2] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile, 2021.

[3] Samyadeep Basu, Xuchen You, and Soheil Feizi. On second-order group influence functions for black-box predictions. 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 715–724. PMLR, 13–18 Jul 2020.

[4] Louis Béthune, Alberto González-Sanz, Franck Mamalet, and Mathieu Serrurier. The many faces of 1-lipschitz neural networks, 2021.

[5] R. Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[7] Mathieu Damour, Florence De Grancey, Christophe Gabreau, Adrien Gauffriau, Jean-Brice Ginestet, Alexandre Hervieu, Thomas Huraux, Claire Pagetti, Ludovic Ponsolle, and Arthur Clavière. Towards certification of a reduced footprint acas-xu system: A hybrid ml-based solution. In Ibrahim Habli, Mark Sujan, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 34–48, Cham, 2021. Springer International Publishing.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] R. Giordano, W. Stephenson, Runjing Liu, Michael I. Jordan, and T. Broderick. A swiss army infinitesimal jackknife. In *AISTATS*, 2019.

[10] Ryan Giordano, Michael I. Jordan, and Tamara Broderick. A higher-order swiss army infinitesimal jackknife, 2019.

[1] www.deel.ai

[11] Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Louis A. Jaeckel. The infinitesimal jackknife. https://faculty.washington.edu/fscholz/Reports/InfinitesimalJackknife.pdf.

[14] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3):598–608, Mar 2019.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[16] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MTSUMMIT*, 2005.

[17] Pang Wei Koh, Kai-Siang Ang, Hubert H. K. Teo, and Percy Liang. On the accuracy of influence functions for measuring group effects, 2019.

[18] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[19] Zhifeng Kong and Kamalika Chaudhuri. Understanding instance-based interpretability of variational auto-encoders, 2021.

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[21] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[23] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[24] Vardan Papyan, X. Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, Sep 2020.

[25] Garima Pruthi, Frederick Liu, Mukund Sundararajan, and Satyen Kale. Estimating training data influence by tracking gradient descent. *CoRR*, abs/2002.08484, 2020.

[26] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In David Helmbold and Bob Williamson, editors, *Computational Learning Theory*, pages 416–426, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[29] Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[30] Yi Sui, Ga Wu, and Scott Sanner. Representer point selection via local jacobian expansion for post-hoc classifier explanation of deep neural networks and ensemble models. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[31] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[32] Chih-Kuan Yeh, Joon Sik Kim, Ian En-Hsu Yen, and Pradeep Ravikumar. Representer point selection for explaining deep neural networks. *CoRR*, abs/1811.09720, 2018.