Internship report

Influence functions for global explainability of deep neural networks

DATE: 03/04/2023

Author	*Function & name*	*Guilhem Fouilhé*	*Student*
Advisors	*Functions & names*	*David Vigouroux* *Agustin Martin Picard*	*Research Engineers*
Academic advisor	*Function & name*	*Edouard Pauwels*	*Assistant professor*

Abstract

This research internship within the DEEL project at the IRT Saint-Exupéry focuses on addressing the global explainability of Deep Neural Networks through the lens of training data attribution, and more specifically influence functions. The aim is to identify functioning groups within a dataset, exploring relationships between data points and their collective impact on the model's training. After providing a quick review on training data attributions methods we implemented and have experimented several methods on both a 2D toy dataset and the CIFAR-10 dataset. Although we obtained promising results on the toy model, our results on CIFAR-10 suggest that the task is highly model and initialization dependent, and that functioning groups are not necessarily aligned with natural clusters on the embedding space even for near-duplicates retrieval.

Contents

1 Introduction

1.1 Context of the internship

The second year of the *Mathematics and Computer Science for AI* Master's degree of the *Université Paul Sabatier* is divided in two semesters. Like many degrees of this field, the second one is almost entirely dedicated to a long final internship, which is the last step before graduating. My objective was to choose it with the perspective of getting prepared as much as possible for after my graduation. As i wanted to pursue a PhD program in Machine Learning, a research internship in the very stimulating environment of the DEEL project was a great opportunity.

1.1.1 The IRT Saint-Exupéry

Named after the famous writer and aviator *Antoine de Saint-Exupéry*, the IRT Saint-Exupéry (IRT stands for *Institut de Recherche Technologique*) is a private research foundation supported by the french government, which funds projects in proportion to the industrial contribution and defines the regulatory framework of the foundation [**IRTabout**]. It is currently based in the Innovation Center B612 (in reference to the origin planet of *The Little Prince* of *Antoine de Saint-Exupéry*), as well as several others local actors of technological research (Aerospace Valley, ANITI, Airbus OneWeb Satellites, ...).

The IRT's technological strategy is defined in 8 positioning of Excellence :

- High-value added materials
- Digital tools for materials and structures
- Electrical insulation systems
- Electronic power conversion
- Embedded Artificial Intelligence
- Non-terrestrial networks (6G)
- Digital design optimization
- Digital twins

The DEEL project, that i was part of, has its place in the "Embedded Artificial Intelligence" positioning. This Artificial Intelligence (AI), frugal and certifiable, will be integrated into future distributed, sovereign, and secure architectures for aeronautics, space, and embedded systems. Traditionally trained in centralized cloud architectures, current AI models such as machine learning are data and computation-intensive, offering only limited guarantees. In order to align with environmental challenges, certification, security, and to be compatible with future distributed architectures, IRT Saint Exupéry is developing new AI models that are more frugal, robust, explainable, and resilient.

1.1.2 The DEEL Project

DEEL (*DEpendable and Explainable Learning*) is a transatlantic (Toulouse - Quebec) research project that involves academic and industrial partners in the development of dependable, robust, explainable and certifiable artificial intelligence technological bricks applied to critical systems. Members include IVADO, CRIAQ, IID in Quebec and ANITI, IRT Saint-Exupéry in Toulouse. [**DEELabout**]

Current scientific challenges tackled are Bias, Out-of-distribution, Explainability, Reinforcement Learning and Theoretical guarantees.

As suggested by the title of my internship, i've joined the "Explainability" team. But my topic also included a Fairness stake, that's why i also integrated the "Biais" challenge.

1.1.3 Contributions

I have independently taken on the research project, maintaining weekly meetings with my supervisors and seeking occasional assistance from fellow DEEL members who were also engaged in projects related to influence functions.

I also personally implemented every experiment on Section 4, taking advantage from the already existing *Influenciae* library when possible. Due to the group API not being totally completed, I had to implement or re-implement some methods using the low-level API of the library. For instance, the computation of the group influence as-is was not capable of taking advantage of our iterative method's property to make it efficient. Similarly, the Weights Boundary method had yet not been adapted to groups.

1.2 Research project of the internship : Identifying functioning groups using data attribution methods

Given a model trained on a dataset, an interesting question to ask is : **"Are there groups of samples that share similarities ?"**

We will specifically address this question from the point of view of the impact on the training of the model. Our objective is to be able to tell which examples had a similar impact on the training of the model (or, from another perspective, which examples have been treated similarly by the model during its training). We are investigating methods that have the potential of dividing the training dataset in *functioning groups*, which are such groups of samples.

To understand what we here call a functioning group, consider the following example. Given a model that predicts an emotion based on an image of a face, it's not hard to imagine that, even if this is not the final task of the model, it may internally discriminate natural groups among all the faces (e.g. feminine/masculine appearance, skin color, accessories worn, ...) and treat them similarly. In this example, the group of all feminine looking faces, as well as the group of white skin faces or eyeglasses wearers, can all be considered as functioning groups.

The information that a given model has such a functioning can be very valuable for interpretability. The functioning group that is related to an input can be considered as a *meaningful explanation* and can also be a useful measure of the *Fairness* of our model. This also can have applications in dataset distillation.

An obvious way of tackling this problem is clustering on an intermediate representation (embedding space) of the model. This usually gives clusters that share visual or semantical similarity and retrieves well almost-duplicates in the dataset. This is also a useful insight of a model because it allows to study the representation that then model learnt. However, we want to tackle the problem from the perspective of data attribution, and common impact on the model during training phase.

Whether (and to what extent) clusters of the embedding space are similar to what we call *functioning groups* is a difficult question that we tried to study during this internship.

1.3 Experimental setup and implementation

1.3.1 Software

The DEEL team developed a few months before my arrival a TensorFlow Library: *Influenciae* that implemented most data attribution methods I used. This guided my choice of working mostly in TensorFlow-based python environments.

I stored all my code on a private GitHub repository that allowed me to work from multiple devices without having to worry about transferring files.

I also tried to make my code *as clean, documented, reproducible as possible* and to fit professional standards, in order to be able to share my work easily and to get used to working in these conditions.

I used *Weights & Biases* to keep track of my experiments and to debug problems related to memory which was often a problem.

1.3.2 Hardware

I luckily had access to a 15 machines (with different GPU capacities) cluster dedicated only to the DEEL project. All my code ran on these machines.

1.3.3 Models used

We will apply proposed methods on some models :

- **MLP** is used only for our 2D toy dataset 4.1. It is composed of three fully connected layers of $32, 16$ and 1 neurons with $ReLu$ activation functions for hidden layers, and $sigmoid$ for the last one, to solve the binary classification problem. We eventually added $dropout$ after the two hidden layers for some experiments. It was sufficient to solve the classification task with ≥ 0.95 test accuracy and was easy enough to train to compute *leave-one-out influence* quickly.
- **CNN** a very trivial model to solve classification on CIFAR-10 task 4.2. The architecture consists of three convolutional layers with $32, 64$, and 64 filters, respectively, each followed by a ReLU activation function and a a $2x2$ max-pooling layer, and a classification block that flattens the output then maps it to a fully connected layer of 64 neurons before the final layer that contains 10 neurons.
- **ResNet** which is Resnet18[**he2016deep**] as implemented in the benchmark module of *influenciae* [1].
- **VGG19** as implemented in *tensorflow.keras.applications*

1.4 Notations

Across all this text, we will consider a training dataset $\mathcal{S} = \{z_1, ...z_n\} \subset \mathbb{R}^{n \times p}$ where $\forall i = 1, ..., n \;\; z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ on which we train a model of parameters $\theta \in \Theta \subset \mathbb{R}^k$ with a loss function $\mathcal{L} : \mathbb{R}^p \times \Theta \longrightarrow \mathbb{R}^+$.

[1] https://github.com/deel-ai/influenciae/blob/main/deel/influenciae/benchmark/model_resnet.py, with default settings and default data augmentation and training settings.

2 Related work

2.1 Explainable Artificial Intelligence and Training Data Influence Analysis

Deep neural networks (DNN) are a class of Machine Learning (ML) models that have achieved state of the art performance in many applications including medicine, transportation, security and finance. It thus already have broad societal implications that are likely to grow during the next decades.

To a certain extent, the behavior of a ML model is transparent to its developer : any prediction consists on a known sequence of basic mathematical operations that are reproducible provided sufficient information about the model (architecture, weights, random initialization ...). However, this transparency is useless when the goal is to capture the "reasoning" behind a prediction, which is something a user can be interested to have access to when he is impacted by that prediction, or when a decision based on the prediction of the model can have important consequences.

It is indeed in some cases possible to have access to an "explanation" that is both not-too-far from the actual model's behavior and acceptable by the user (e.g. a human being). This is actually an important field of research currently in Machine Learning named *Explainable Artificial Intelligence* (XAI) (see [**SurveyXAI**] for a detailed review). Being able to provide such an explanation will be mandatory in some case in the frame of the "right to explanation" that European laws intend to guarantee to its citizens[2].

2.1.1 Training Data Influence Analysis

Among the vast literature in XAI, we will focus on *(training) Data Attribution* methods (see [**Hammoudeh2022Dec**] for a detailed review), i.e. the task of tracing model predictions back to the training examples that induced the model into making these predictions.

Influence functions are a class of these methods that we will be particularly focusing on. We will define them more in detail/more precisely in Section 2. The main idea is to use a Taylor expansion to approximate an infinitesimal up-weighting of a training point in the loss function.

2.1.2 Local vs global explanation

XAI tools can be divided into two main categories: Local and Global, each serving the purpose of explaining how AI models generate predictions at different levels. The Local methods focus on understanding the behavior of AI algorithms on specific situations. These methods often rely on a single observation or a small subset of observations to provide metrics that reveal the contribution of each feature to the final prediction made by the AI model. They are particularly useful for individuals who are affected by the algorithm's decisions; for instance, a lending company may need to explain why a certain customer's mortgage request was denied by their AI model [**Bracke2019Aug**]. Local XAI methods are prevalent in recent literature concerning individual and counterfactual fairness and play a crucial role in developing and enhancing these concepts [**Ge2022Jul**, **Albini2022Jun**].

On the other hand, the Global set of XAI tools focuses on understanding AI algorithms' behavior at a higher level. These tools aim to help users measure how features contribute to predictions across the entire dataset or a collection of datasets. They are typically used by AI algorithm developers, regulators, AI managers, and those involved in scientific discoveries. These users are more interested in gaining general insights and knowledge discovery from the AI model, rather than focusing on specific individual observations.

In this work we are mostly interested in global explainability.

[2] Recital 71, EU GDPR

2.2 Leave-one-out influence

Our approach of the training data attribution problem is ideally solved by the simple and interpretable concept of *Leave-one-out influence (LOO)*, a tool coming from robust statistics [**Cook1982**, **hampel1974influence**].

Definition 2.1 (Leave-one-out influence)**.** The *Leave-one-out influence* of a training sample z on another training sample z_{test} is defined as :

$$LOO(z, z_{test}) = \mathcal{L}(z_{test}, \theta^{(T)}_{\mathcal{D}\setminus z}) - \mathcal{L}(z_{test}, \theta^{(T)}_{\mathcal{D}}) \tag{1}$$

Where $\mathcal{L}(z, \theta^{(T)}_{\mathcal{D}})$ is the loss of the model of parameters θ after T training steps with $\mathcal{D}$ training dataset, evaluated on z. It can be interpreted as *the impact of the removal of the training sample z on the trained model*.

Remark. When $z = z_{test}$ we will write $LOO(z) := LOO(z, z)$ and name it *self leave-one-out influence of z*.

The evaluation of the complete LOO for the entire training set necessitates the training of $(n+1)$ models. It is worth noting that when dealing with a deterministic model class and training algorithm, LOO emerges as one of the limited influence measures that can be precisely computed in polynomial time concerning both the training-set size, n, and the iteration count, T. This characteristic makes LOO a valuable and efficient tool for analyzing the impact of individual data points on the model's performance during training.

The biggest strength of LOO influence lies in its simplicity, making it easily understandable even by non-experts. This simplicity has led to its application in ensuring fairness in algorithmic decisions. LOO can be used with any model architecture due to its straightforward nature. However, the main drawback is its high computational cost, particularly when dealing with modern large-scale datasets and complex models. Training multiple models for LOO estimation is impractical for all but the largest organizations, and it also has environmental implications. Additionally, modern models have an important predictive variance that makes it difficult to disentangle instance deletion effects from intrinsic variability [**Feldman2020Aug**], further complicating LOO analysis. Despite these challenges, LOO's impact on influence analysis research is significant, as many related methods either estimate LOO influence directly or share similarities with it.

Remark. Analogously, we can define *Leave-k-out influence* as the impact of removing k points from the training dataset.

2.3 Influence functions

2.3.1 Definition

Among influence estimators, Influence Functions (IF) [**Koh2017Mar**] are one of the most well-known and studied. These estimators are derived from influence functions (or *infinitesimal jackknife*) in robust statistics [**hampel1974influence**], which analyze how models change when a training instance's weight is infinitesimally perturbed.

More specifically, instead of directly considering the impact of the removal of a point, we can look at the more general problem of the impact of *up-weighting* it. To define it we consider that we train a model of parameters θ through *empirical risk minimization* :

$$\theta^* = argmin_\theta \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(z_i, \theta)$$

We assume that the empirical risk is twice-differentiable and strictly convex in θ.

Up-weighting by ϵ the training sample $z \in \{z_1, \ldots z_n\}$ means targeting the model :

$$\theta_z^\epsilon = argmin_\theta \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(z_i, \theta) + \epsilon \mathcal{L}(z, \theta)$$

Remark. Up-weighting z by $\epsilon = -1/n$ is equivalent to removing z from the training dataset.

The main idea of influence functions [**Koh2017Mar**] is to estimate θ_z^ϵ by its first order Taylor series expansion :

$$\theta_z^\epsilon \approx \theta^* - \epsilon H_{\theta^*}^{-1} \nabla_\theta \mathcal{L}(z, \theta^*)$$

Where $H_\theta = \frac{1}{n} \sum_{i=1}^{n} \nabla^2 \mathcal{L}(z, \theta)$ the Hessian of the model, positive definite (PD) by the strict-convexity assumption and symmetric by the Schwarz's theorem.

This uses a classical result [**Cook1980Nov**] that shows that we can write the change in the model parameters as follows :

$$\frac{d\theta_z^\epsilon}{d\epsilon}\Big|_{\epsilon=0} = -H_{\theta^*}^{-1} \nabla_\theta \mathcal{L}(z, \theta^*)$$

This already provide a metric of influence of z by directly estimating the change in the model parameters. This is typically a large vector (size k, the number of parameters) that we will name **Influence vector** of z.

Definition 2.2 (Influence vector)**.** The *Influence vector* of a training sample z on model parameterized by θ is the value :

$$\mathbf{I}(z) = H_{\theta^*}^{-1} \nabla_\theta \mathcal{L}(z, \theta^*)$$

By using chain rule we can measure the change of any differentiable function of θ. In particular, there is an easy expression for the change on the loss at a test point z_{test} :

Definition 2.3 (Influence value)**.** The *influence* of the training sample z on the loss at a test point z_{test} is defined as:

$$\mathcal{I}(z, z_{test}) = -\nabla_\theta \mathcal{L}(z_{test}, \theta^*)^T \mathbf{I}(z) = -\nabla_\theta \mathcal{L}(z_{test}, \theta^*)^T H_{\theta^*}^{-1} \nabla_\theta \mathcal{L}(z, \theta^*)$$

2.3.2 Efficient implementation

Computing $\mathcal{I}(z, z_{test}) = -\nabla_\theta \mathcal{L}(z_{test}, \theta^*)^T H_{\theta^*}^{-1} \nabla_\theta \mathcal{L}(z, \theta^*)$ raises computational challenges. Computing $H_{\theta^*}^{-1}$ requires $O(nk^2 + k^3)$ operations, which is intractable for models with millions of parameters, both in time and in memory. This is a well-studied problem that can be addressed by avoiding computing $H_{\theta^*}^{-1}$ explicitly. Instead, we can approximate implicitly the influence vector $H_{\theta^*}^{-1} \nabla_\theta \mathcal{L}(z_{test}, \theta^*)$ (this is called an Hessian vector product (HVP) [**Pearlmutter1994Jan**]).

In practice, when possible, it is still faster to compute the inverse Hessian explicitly because it only needs to be done once for a model, while HVP solves an optimisation problem for each influence computation. To make this possible we considered only the neural network's last layer containing trainable weights, thus both reducing the size of the hessian and verifying the convexity hypothesis we need for accurate calculation of these quantities. However, the memory requirements of storing the hessian is still a problem and in the cases where it is too large we used conjugate gradient descent [**Steihaug**] to approximate HVP.

2.3.3 Properties and limitations

IF accurately approximates retraining when loss is convex [**Koh2017Mar**]. However, when it isn't, which is more common in modern deep learning models that requires interpretability, the quality of this approximation is fragile . More specifically, it is very dependent on initialization, training procedure (weight decay,...), model architecture (depth and width,...), and choice of test points. The quality of this approximation becomes very poor on large size datasets like ImageNet [**basu2021influence**].

Although these results seems to indicate that IF may not be informative in deep learning settings, more recent research showed that IF were indeed good approximators of another object called *proximal Bregman response function (PBRF)* [**bae2022if**], which approximates the effect of removing a data point while trying to keep the predictions consistent with those of the (partially) trained model. The PBRF doesn't necessarily correspond to the LOO due to warm-start gap (caused by the assumption of approximating training starting from parameters that are close to the optimum), the proximity gap (caused by the addition of an L2 regularization typically used) and the non-convergence gap (caused by stopping the training procedure not exactly when the gradient of the loss is 0). The only gap remaining between IF and PBRF is the linearization gap, which remains bounded even in realistic Deep Learning settings. Crucially, the existing gap between LOO and PBRF doesn't prevent from replacing LOO by PBRF in most applications (including identifying influential or mislabelled examples) since these use-cases don't rely on exact LOO retraining.

Similarly, another even more recent study [**schioppa2023theoretical**] concludes the usage of a Taylor expansion despite conditions not holding is responsible for IF methods being time-bound, meaning the most they can do is predict a model's training on a few steps, not on the entire retraining, even in deterministic settings.

In conclusion, although IF are poor when evaluated on their capacity to approximate LOO in realistic settings, this isn't an obstacle for motivational uses cases and the main idea of IF - linearization - doesn't make a important difference in the approximation of LOO and PBRF. Thus we can still consider IF as an informative and computationally useful method and consider its use-cases.

2.4 Influence functions for groups of points

In 2.3 we have defined the influence of a single sample on another single sample. Although this opens a lot of applications including data cleaning [**picard:hal-03617649**] or prediction explanations [**Koh2017Mar**], the insights about the model that we can obtain from these is limited as a typical model ML model is typically very weakly impacted by a single sample. To address this and leverage IF for global explainability, we need the notion of *Group influence* introduced in [**Koh2019May**] as an approximation of *leave-k-out influence*.

We represent a group of samples by a binary vector $\mathbf{w} \in \mathbb{R}^n$ where $w_i = 1$ if the training sample $z_i \in \mathcal{S}$ is included in the group, and 0 otherwise. The parameters obtained after training the model on $\mathcal{S}$ is $\theta^*(\mathbb{1})$, and the obtained parameters after training the model on every sample of $\mathcal{S}$ but $\mathbf{w}$ is $\theta^*(\mathbb{1} - \mathbf{w})$.

Definition 2.4 (Leave-k-out influence)**.** The *Leave-k-out influence* (or leave-some-out) of group of training sample $\mathbf{w}$ on another group of training sample $\mathbf{w}_{test}$ is defined as :

$$LOO(\mathbf{w}, \mathbf{w}_{test}) = \mathcal{L}(\mathbf{w}_{test}, \theta^*(\mathbb{1} - \mathbf{w})) - \mathcal{L}(\mathbf{w}_{test}, \theta^*(\mathbb{1})) \tag{2}$$

Note that we make no hypothesis on the size of both groups.

We can then introduce group influence as an approximation of leave-k-out influence, or more precisely according to 2.3.3 an approximation of PBRF.

Definition 2.5. The *influence* of the group of training samples $\mathbf{w}$ on the group of training samples $\mathbf{w}_{test}$ is defined as:

$$\mathcal{I}(\mathbf{w}, \mathbf{w}_{test}) = (\sum_{i=0}^{n} w_i \nabla_\theta \mathcal{L}(z_i, \theta^*(\mathbb{1})))^T H^{-1}_{\theta^*(\mathbb{1})} (\sum_{i=0}^{n} w_{test_i} \nabla_\theta \mathcal{L}(z_i, \theta^*(\mathbb{1})))$$

The authors of [**Koh2019May**] showed that group influence accurately predicted leave-k-out influence in convex settings like a logistic regression. An attempt to use them in realistic deep learning settings and compare them to second order influence functions (which are just influence functions with a second order Taylor expansion instead of first order, and gives marginal approximation improvement in exchange for a larger computational cost) showed that the correlation with LOO was unsurprisingly poor [**basu2020second**].

2.5 Other Data attributions methods

2.5.1 Datamodels

The "datamodelling" framework [**Ilyas2022Feb**] is a very powerful general-purpose data attribution method that builds an explicit model for predictions in terms of the training data.

For any function $f : f(z, \mathcal{S}) = \textit{the result of training a model on dataset } \mathcal{S} \textit{ and evaluating it on } z$, the datamodelling framework estimates, given a distribution over all possible training datasets $\mathcal{D}_\mathcal{S}$ and a target point z, a parametric function $g_\theta : \{0,1\}^{|\mathcal{S}|} \longrightarrow \mathbb{R}$ with

$$\theta = argmin_w \hat{\mathbb{E}}^{(m)}_{S_i \sim \mathcal{D}_\mathcal{S}} [\mathcal{L}(g_w(S_i), f(z, S_i))]$$

Where $\hat{\mathbb{E}}^{(m)}_{S_i \sim \mathcal{D}_\mathcal{S}}$ is a $m - sample$ estimation of the expectation.

The authors of [**Ilyas2022Feb**] found that *Linear datamodels* (where $g_\theta(S) = \theta^T \mathbb{1}_S + \theta_0$) were sufficient to accurately predict predictions of a model from its training data. As other data attribution methods, datamodels can be useful to interpret predictions by finding samples from the training set that are the most semantically similar to the ones under study (Figure 1).

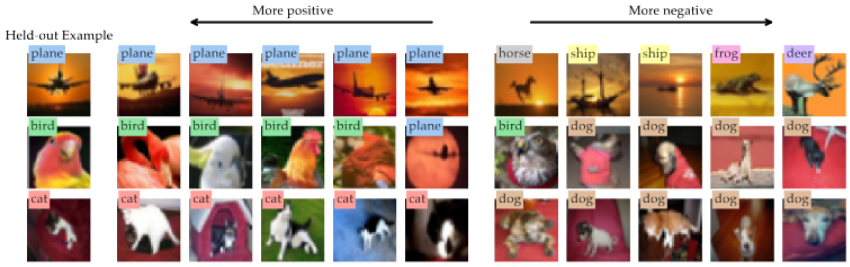


*Figure 1 – Large datamodel weights correspond to similar images. Figure 9 from [**Ilyas2022Feb**]*

The key of the versatility of datamodels relies on the embedding space it yields. To each sample $z \in \mathbb{R}^p$ is associated a datamodel of parameters $\theta \in \mathbb{R}^n$. We can use that vector as an *embedding representation* of z (Figure 2).

That point of view allows a *global* study of the model from the *training data attribution* perspective, which is very close to the target of the internship. However, the counterpart of the power of datamodels is that they are computationally very expensive, as we need to retrain models several hundreds of thousands times to reproduce their results on CIFAR-10. We thus choosed to not use datamodels directly, but we tried an approach using *influence functions* to approximate datamodels in 4.1.

*Figure 2 – Principal components of the embedding space yielded by datamodels can show interesting insights about functioning groups of the model. Figure 14 from [***Ilyas2022Feb***]*

2.5.2 TRAK

TRAK (Tracing with the Randomly-projected After Kernel) [**Park2023Mar**] has the ambition of being a computationally tractable attribution method that approximates retraining almost as well as linear datamodels. It relies on the following steps :

1. Linearizing the model's output function (via Taylor approximation), which reduces the model of interest to a linear function in parameter space.
2. Reducing the dimensionality of the linearized model using *random projections* [**MR0737400**].
3. Estimating attribution scores using a leveraged version of *One-step newton approximation* [**Pregibon1981Jul**] (the authors showed that some terms were insignificant in term of approximation quality while being computationally expansive and thus dropped these terms in their implementation).
4. Ensembling results over several models, each trained on a random subset of the original training dataset S
5. Sparsifying the attribution scores using soft-thresholding.

The authors showed that this method achieved a *state-of-the-art* trade-off between efficiency and accuracy. These results are summarized in Figure 3.

The pre-print of this paper was released just at the beginning of the internship. We therefore started to hear about its existence after starting to work with influence functions. This method could have been a good alternative to anything that we solved with IF but because it was still recent the only public implementation was in Pytorch, as opposed to all other implementations of methods used that were in Tensorflow. That's why we decided not to study this method further during this internship. Nevertheless, we believe that studying this method and its applications could make for a very interesting research project.

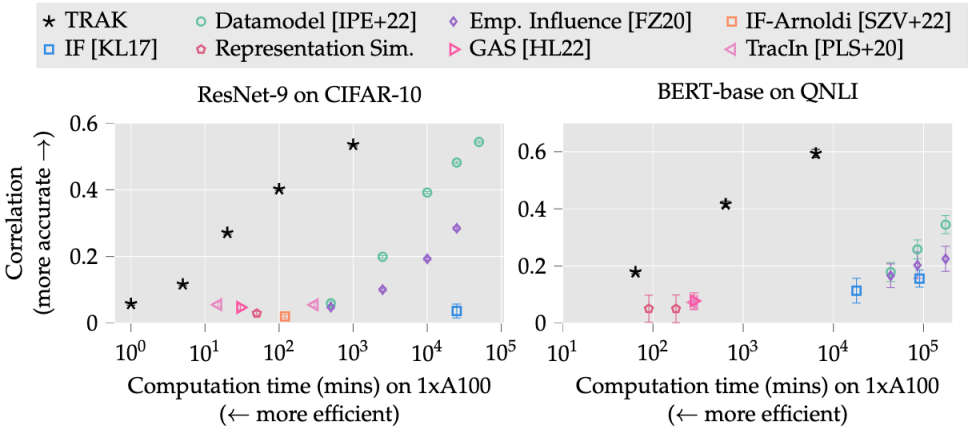


*Figure 3 – TRAK achieved state-of-the art trade-off between efficiency and accuracy. Figure 1 from [**Park2023Mar**]*

3 Rigorous definition of objectives

3.1 Functioning Group

Definition 3.1 (Functioning Group)**.** We call a *Functioning Group* over possible subsets $\mathcal{P} \subset P(\mathcal{X})$ a subset $\mathcal{G} \in \mathcal{P}$ that maximizes the function :

$$\mathcal{P} \longrightarrow \mathbb{R}$$
$$\mathcal{S} \longmapsto \frac{\mathcal{I}(\mathcal{S})}{\#\mathcal{S}}$$

Remark. In practice, a useful class of Functioning groups are constructed with $\mathcal{P}$ of the form

$$\{\mathcal{S} \text{ s.t. } x_1, \dots x_m \in \mathcal{S} \text{ and } x'_1, \dots, x'_{m'} \notin \mathcal{S}\}$$

We're interested in finding a partition of $\mathcal{X}$: $\mathcal{G}_1, \dots \mathcal{G}_k$, i.e. a mutually exclusive and jointly exhaustive division of $\mathcal{X}$.

3.2 Iterative methods for functioning group detection

Given a dataset $\mathcal{X}$ and a model,

Definition 3.2. We call an *Iterative Method* a greedy method for constructing a *Functioning group*. More precisely, we construct a group by starting with a given sample (or group of sample) as a seed (x_0) and for $t = 1, \dots$ we add to the group $\{x_0, \dots x_{t-1}\}$ the sample x_t that maximizes the group influence, which is the group we construct using $\mathcal{P} = \{\mathcal{S} \text{ s.t. } x_0, \dots x_{t-1} \in \mathcal{S}\}$ with notations of Definition 3.1.

Algorithm 1 Illustration of an Iterative method for group detection

Require: $x_0 \in \mathbb{R}^p$, $Dataset\ \mathcal{S}$, $stopcond$
  $\mathcal{G} \leftarrow \{x_0\}$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus x_0$
  while $not\ \ stopcond$ **do**
    $I \leftarrow \{\}$
    for $x \in \mathcal{S}$ **do**
      $I \leftarrow I + \mathcal{I}(x, \mathcal{G})$ ▷ $\mathcal{I}(x, \mathcal{G})$: normalized influence of x on $\mathcal{G}$
    end for
    $x_{new} \leftarrow argmax(I)$ ▷ $l + x$: append x to list l
    $\mathcal{G} \leftarrow \mathcal{G} + x_{new}$
    $\mathcal{S} \leftarrow \mathcal{S} \setminus x_{new}$
  end while
  return $\mathcal{G}$

4 Experiments

Proceeding to develop a method that maps every sample in the dataset to its related functioning group (Definition 3.1) is an ambitious project. The scope of this internship can be seen as a preliminary work, in the sense that we investigate whether or not this is possible by leveraging currently existing data attribution methods for groups. For this, we essentially applied our iterative method to a two-dimensional synthetic dataset (Subsection 4.1) and CIFAR-10 (Subsection 4.2).

4.1 Results on a 2D toy dataset

To validate the method we started on a two-dimensional synthetic dataset and two classes on which points of one class (the red one) are separated in two clusters. We expect our model to treat the clusters as two distinct *functioning groups* (Figure 4).

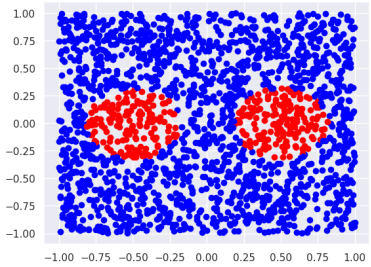


Figure 4 – Our 2D toy dataset. The red points are labelled 1 *and the blue points are labeled* 0*. The input is simply the 2-dimensional position of the point.*

4.1.1 Most influential points

As expected, according to previous research [**picard:hal-03617649**], the most influential points in term of *self-influence* are those located at the boundary of decision. This property reveals very useful for dataset cleaning and already provides valuable insights about our model (Figure 5).

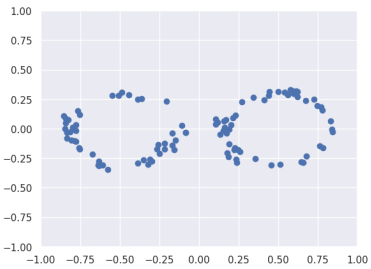


Figure 5 – The top-100 points (in term of self-influence) of the dataset

However, self-influence of single samples fails to capture interactions between them. One possibility offered by influence functions is to look at at the influence of a sample with respect to another. When computing this score for the centers of the clusters, the results are very model-dependant: model architecture, optimizer, learning rate and regularization can have an important impact. In Figure 6, we show the impact of the regularization on this score. If our goal is to focus on interactions and not global influence, we're looking for a model setup that promotes the points that are in the same cluster. We used the results of this experiment to find a model that would be likely to identify the two clusters as two *functioning groups*.

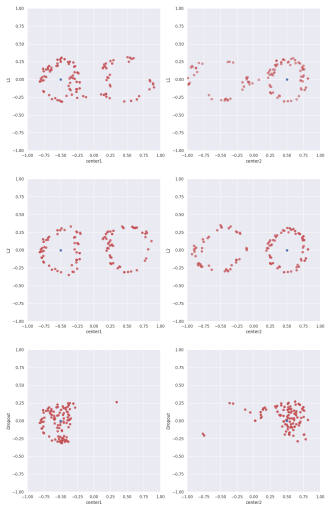


Figure 6 – The top-100 points (in term of influence relative to the blue point) of the dataset for $L1, L2$ and $Dropout$ regularizations. In this visualization, we see that Dropout regularization seems to promote local interactions over global influence. This is intended to demonstrate that the notion of influential points is highly model-dependant, which is both good news because it highlights the potential of global explainability of our method, and bad news because it means finding a method that works for any model is difficult.

4.1.2 Evolution of the influence with the number of points

In 7 we investigate (as a sanity check of the influence metric for groups), that scores grow with size in any configuration and that a specific group is more influential than a group of the same cardinal chosen at random.

It indeed has the expected behavior as demonstrated by the fact that : $\mathcal{I}(random\_boundary\_left) \geq \mathcal{I}(random\_left) \geq \mathcal{I}(random) \geq \mathcal{I}(full\_random)$

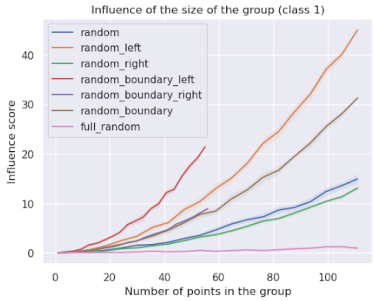


Figure 7 – Growth of the influence score of a group with its cardinal. random: points on the left or right cluster, random(left,right): points on the (left,right) cluster, random boundary (left,right): points on the (left,right) cluster close to the boundary, random boundary: points on the left or right cluster close to the boundary, full random: points on any class.

4.1.3 Iterative method results

We applied our iterative clustering algorithm to this dataset with the center of the left circle as a seed and obtained the results in Figure 8.

In Figure 8, we see that after entirely filling one cluster, our algorithm starts adding points that belong to the other cluster. We don't want that, as our goal was to capture a single cluster. We need to find a stop condition that correspond to the end of a cluster.

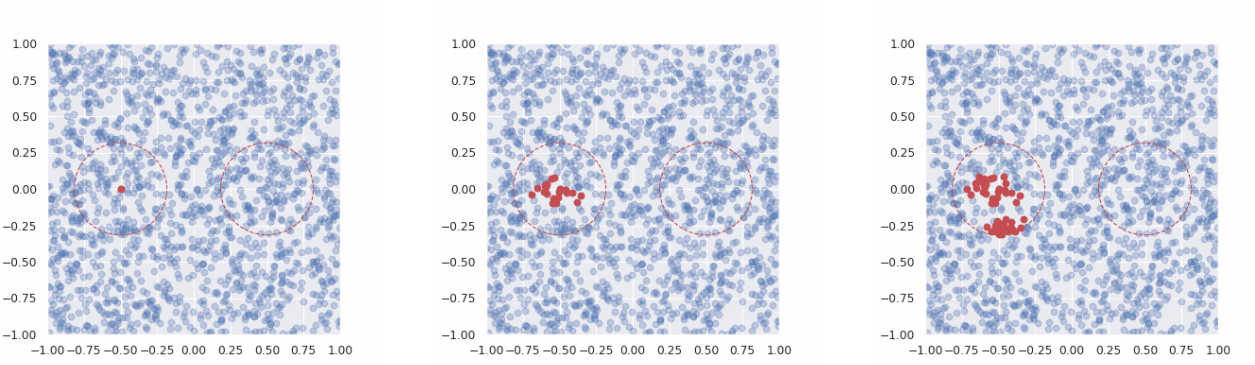
For this, we propose the mode change of influence of the group normalized by its cardinal, which showed to have the expected behavior in our experiments (Figure 9).

However, applying this stop condition to our clustering algorithm doesn't work as well because of the greedy strategy: on the first iterations the group obtained has a much larger influence than a group of points of the same cluster taken randomly of the same size. But when the size of the group gets closer to the size of the cluster these two values must match. That's why the influence of the group must decreases at some point before exhausting the elements of the first cluster. We didn't attempt to find another criterion during this internship as this limitation seemed mostly related to the greedy nature of the strategy.

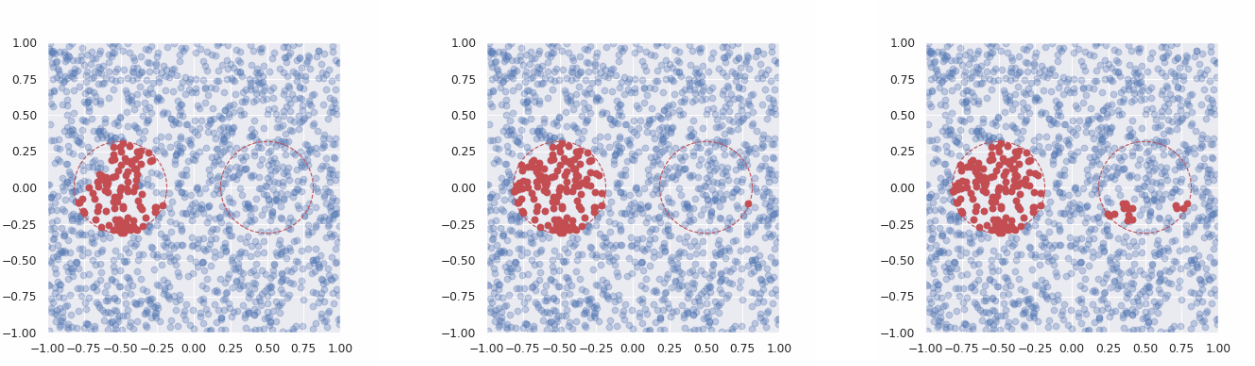
4.1.4 Datamodels approximation

We have seen in 2 that we could leverage datamodels to create a powerful embedding space of training samples that represented their impact on the training of the model. This space could be very useful to our research project. However, the computational cost as-is is prohibitive for realistic applications: the original authors[**Ilyas2022Feb**] used 9 NVIDIA A100 to train 300.000 to 1.5M models (depending on the configuration), just for the CIFAR-10 dataset. We investigated accelerating this computation from the point of view of just replacing each retraining by its first order approximation using influence functions.

A methodical way to fairly evaluate the impact of this approximation would be to compare a metric of interest in the same conditions for both method (retraining and approximation with IF). We didn't do that during this internship because even using pre-computed datamodels of a ResNet on CIFAR-10 that the authors gave access to was costly and non-trivial. Instead, we just wanted to know if *functioning groups* had a chance to be retrieved from a simple clustering on the embedding space induced by the datamodel.



(a) it = 1 *(b) it = 20* *(c) it = 50*



(d) it = 100 *(e) it = 133* *(f) it = 150*

Figure 8 – Clustering at different stages of the iterative process with the center of the left cluster as a seed. The left cluster in entirely exhausted at iteration 132 and thus the following added samples are on the other cluster.

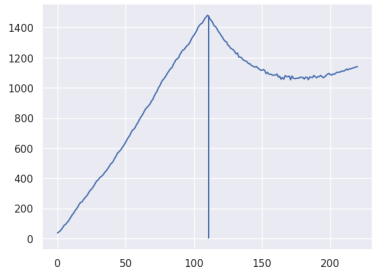
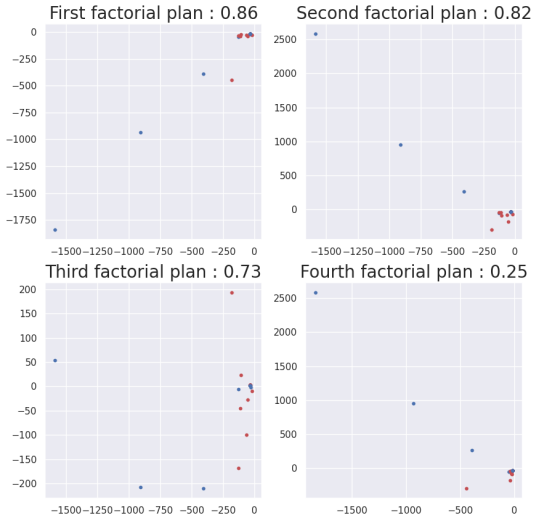
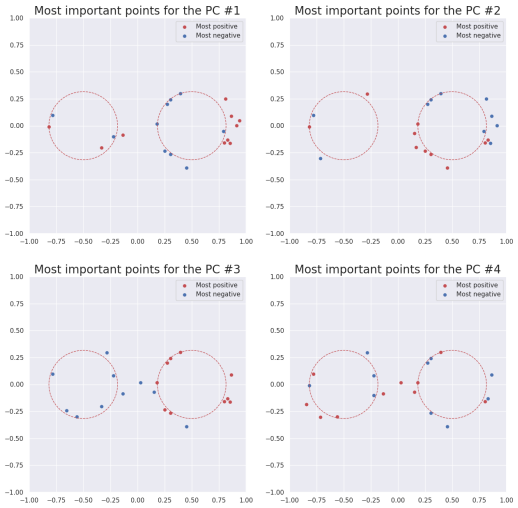


Figure 9 – Influence score of a group normalized by its cardinal when adding first random points from one cluster (before vertical blue line) until there is none left, then random points from the other cluster (after vertical blue line). We clearly see that the moment where the group influence stops increasing is a good condition to determine where to stop the iterative algorithm.

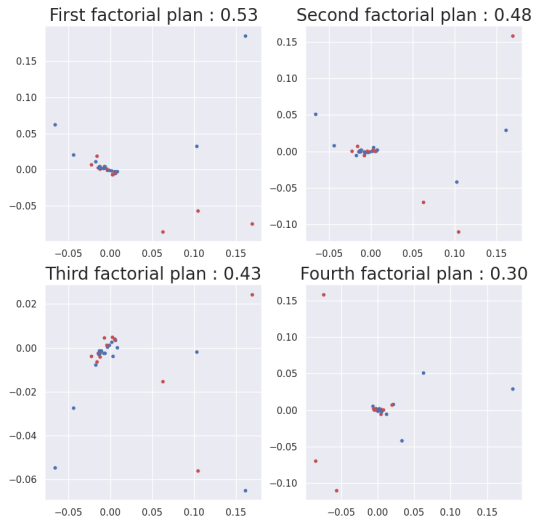
We thus visualized this embedding space using a PCA to see if its structure could be useful for our problem in Figure 10. As we got similar results for various values of $\alpha(0.1, 0.25, 0.5, 0.75, 0.9)$ and $k(50, 100, 200)$ retrainings (of retrainings approximation with IF) for each sample, and couldn't take larger values of k because of computational limitations and memory issues, we didn't investigate further.



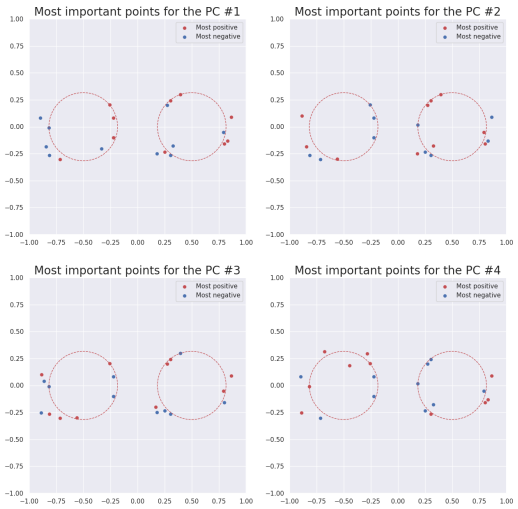
(a) IF (factorial plans)



(b) IF (most influential points)



(c) LOO (factorial plans)



(d) LOO (most influential points)

*Figure 10 – We applied the PCA algorithm on the embedding space obtained for retraining-based and IF-based datamodels. Left: first few factorial plans, blue for left cluster, red for right cluster. Right: display of the 10 most positive and 10 most negative points for the first principal components, similarly to 2. We see that there doesn't seems to be a way to discriminate between the two clusters based on this space. Here α with notations of [**Ilyas2022Feb**] is set to 0.9, which means we are retraining (or approximating retraining) of models with 90% of the dataset.*

4.2 Results on CIFAR-10

The CIFAR-10 (*Canadian Institute for Advanced Research*, version 10) dataset [**Krizhevsky2009LearningML**] is a widely used benchmark dataset in the field of computer vision and machine learning. The dataset consists of 60,000 RGB images, each of size 32x32 pixels, belonging to 10 different classes : Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck.

These images are evenly distributed across the ten classes, with 6,000 images for each class. CIFAR-10 is popularly used for tasks like image classification, object recognition, and deep learning model evaluation. Due to its small image size and diverse classes, it serves as a useful starting point for researchers and students to experiment with various machine learning techniques and algorithms.

4.2.1 Iterative method results

We applied Algorithm 1 to this datasets for various random seeds (Figure 11, we showcase additional results in Appendix A). Results are ambiguous because the visual coherence of functioning groups seems to be very dependent to initialization and model used. This motivated us to investigate the difference between clusters obtained from this method and natural clusters on the embedding space.

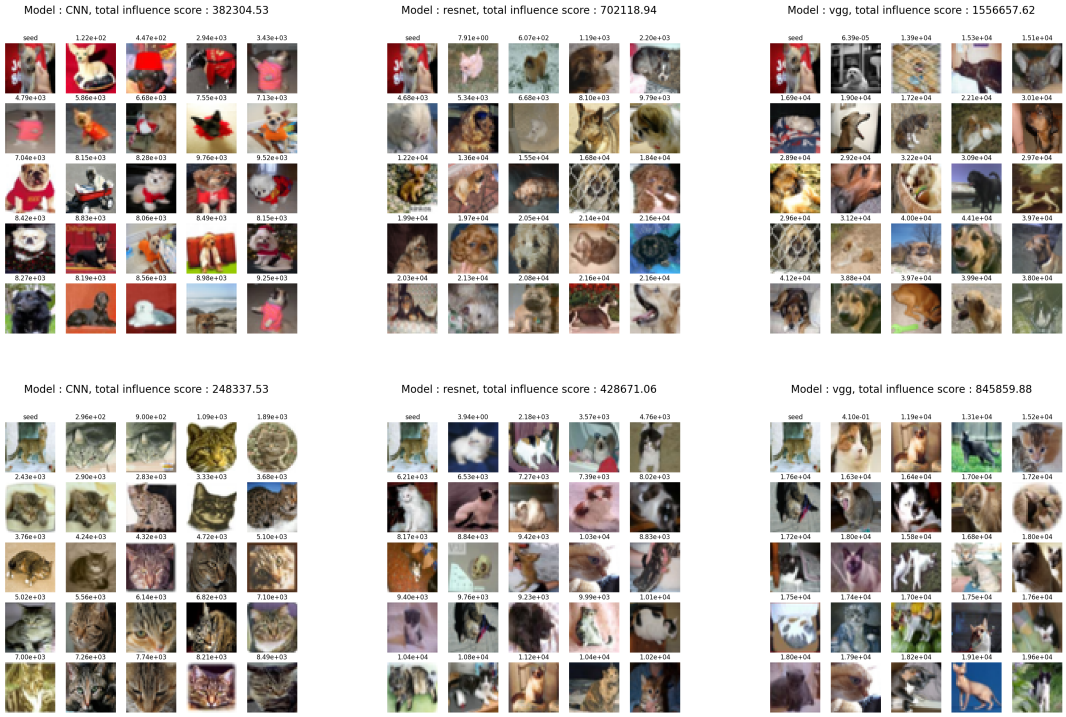


Figure 11 – First 25 images obtained using our iterative method with order 1 influence functions for two different seeds. We see that each model yields very different results. Remarkably, the CNN model (which performs the worst on the classification task) has a much clearer visual coherence between in obtained clusters. We can make the hypothesis that the simple architecture of this model leads to features with a lower level of abstraction that are easier to identify as a human.

4.2.2 Comparison with clustering on the embedding space

An experience we intended as a sanity check of our method was the capacity of retrieval of near-duplicates of the dataset. CIFAR-10 indeed contains a significative number of duplicates and near-duplicates, both inside the train set and in both train and test sets [**barz2020we**]. Here we're only interested in duplicates and near-duplicates in the train set.

Figure 12 – A cherry-picked automobile image from the CIFAR-10 train set known for having near-duplicates in the dataset with clusters obtained by several way. ***(Left)*** *our iterative method (minimizing influence).* ***(Right)*** *our iterative method (maximizing influence), Center: its 25 closest (in $L2$ norm on the features space for the CNN model) in the dataset, with the distance on top of each image. The three clusters are ranked by influence, meaning the center one has no chance of being retrieved by our functioning group detection method.*

As we can see in Figure 12, the near-duplicates are easily retrieved by a simple clustering on the embedding space, but not by our functioning group method. This is not a limitation caused by the greedy iterative method but a limitation of the influence score based approach, as suggested by the fact that there exists both groups with lower and higher influence scores. This causes a major problem as our vision of functioning groups implied that almost identical samples must be in the same functioning group. This motivated our choice of exploring other methods not based in Influence Functions.

4.2.3 Weights Boundary results

The DEEL team that implemented *Influenciae* has also implemented a new method based on a variant of *DeepFool* [**7780651**] currently called *Weights Boundary*.

DeepFool is a prominent adversarial attack method in the field of machine learning. Its primary objective is to generate minimal adversarial perturbations on input data to deceive deep neural networks into making incorrect predictions. The method is based on linear approximation and optimization principles, iteratively projecting input samples onto decision boundaries of deep models to find the smallest perturbation required for misclassification.

DeepFool's strength lies in its simplicity and efficiency, making it a widely used tool for evaluating the robustness of deep learning models. It is agnostic to the underlying architecture, allowing its application to various neural network models.

The *Weights Boundary* method, instead of finding a minimal adversarial perturbation on input data, computes a minimal adversarial perturbation on model's weights. That way we can iteratively deform a model's decision boundary until changing a prediction and define an *influence score* by quantifying the total

perturbation needed to change a prediction, e.g. the norm of the difference between the weights of the model before and after the algorithm (Figure 13).

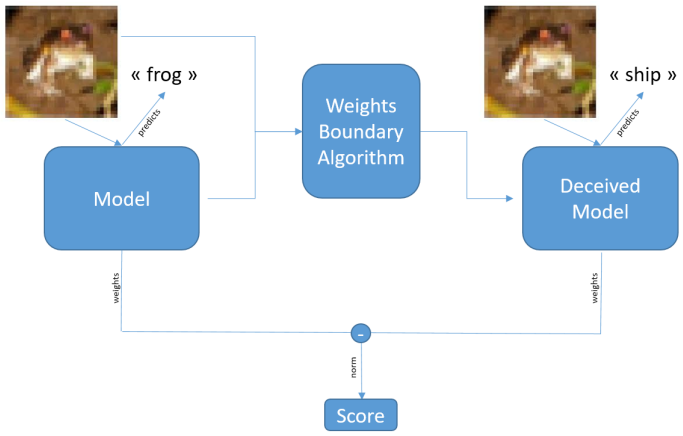


Figure 13 – Visualization of the method to obtain a score from Weights Boundary Algorithm.

In this setting, the *data attribution* problem is reduced to a high dimension optimisation problem.

It is also worth noting that we cannot define a notion of influence relatively to another sample, as in Definition 2.1 or 2.3. Our intuition, however, is that moving the decision boundary to change predictions for a sample may also change (or get closer to) change predictions for samples that are in the same functioning group, more than unrelated samples. Therefore minimizing the score obtained with the Weights Boundary Algorithm could be interesting.

To extend this algorithm to group detection, we need to compute a deformation of the model that can change the model's prediction on the whole group. To do so, we identified two methods:

1. Treating the input group as a batch and change the model in the same way we would just train a model on a single batch.

2. Taking inspiration from Universal Adversarial Attacks [**moosavi2017universal**], changing prediction for each sample individually iteratively without resetting the model weights between elements of the group.

Whereas the second method appears more supported by the literature and needed less adaptation of the currently implemented method in influenciae, we initially used the first one and haven't yet switched to the second one. This is a direction of future work for the few weeks left of internship after the writing of this report.

In our first attempt to naively adapt the DeepFool-based Weights Boundary method for groups, we encountered optimisation issues way too often and too randomly to have intepretable results. Therefore we decided to implement a simpler version where, instead of computing the orthogonal projection on the decision boundary like with DeepFool-based method, we retrained the model with the opposite objective as the one it was trained on until changing the prediction. More precisely, we maximized the cross entropy loss we were maximizing when training our model for the classification task. This was an easier optimisation to solve and we assumed it would give us an easier first step for a proof-of-concept that we could optimize later.

For this method that we call *Simple Weights Boundary*, we used the same optimizer that we used for training with the learning rate at the final step of the schedule. When the deformation of the model got too important (norm of the difference of weights > 1) we or when the optimization took too many steps (> 100), we stopped the optimisation process, considering that this cannot be the case in a functioning group. We show the results of applying the iterative functioning group detection algorithm with *Simple Weights Boundary* in Figure 14. We see that on the VGG model with the dog seed, the sample selected by the algorithm in some cases has a score of 1, meaning at this step the optimization process hasn't worked for any sample and that our algorithm chose a picture randomly on the dataset. This suggests that our simple method encounters a lot of optimisation issues that should be addressable with a more adapted approach. This limitation was expected and addressing it should be a significant part of the rest of the internship.

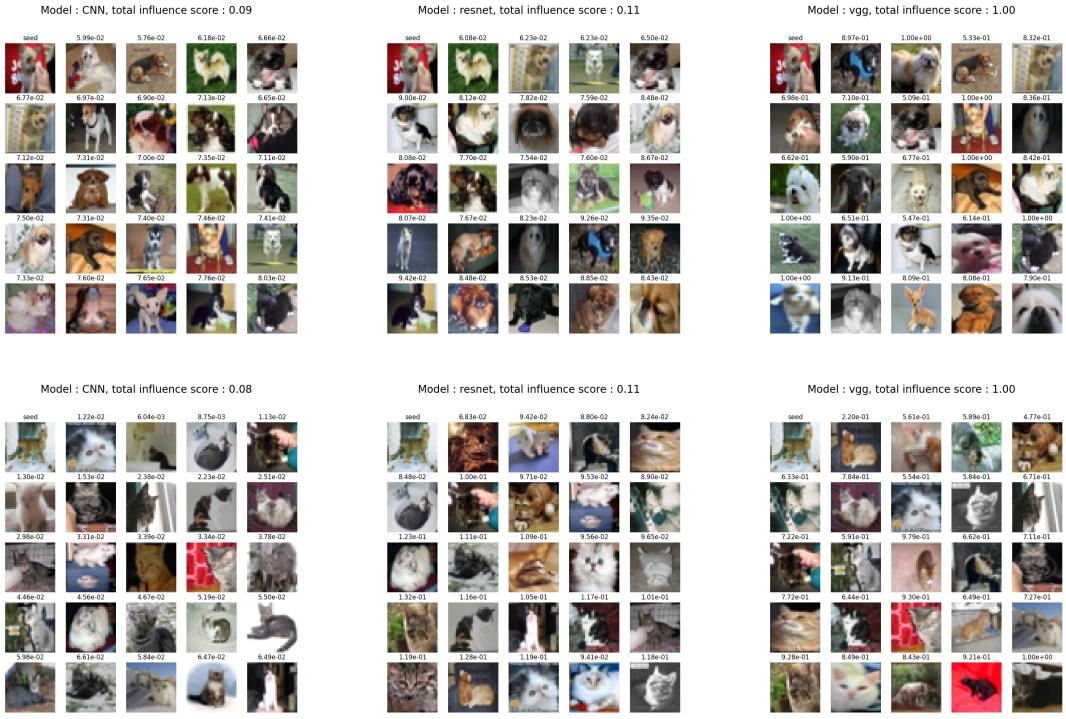


Figure 14 – First 25 images obtained using iterative method with Simple Weights Boundary for two different seeds.

We also found that the success of the optimisation method was highly dependent on the class of the seed (Figure 15).

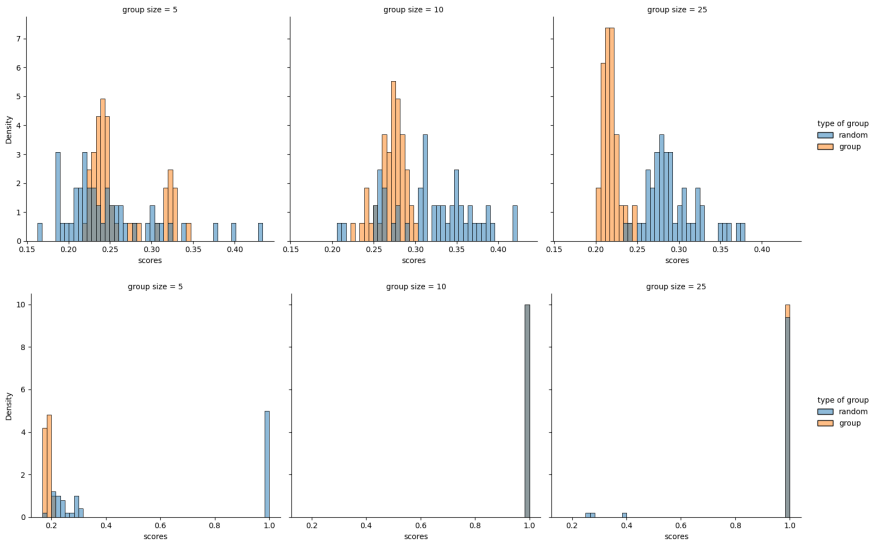


Figure 15 – Score obtained with the Simple Weights Boundary Algorithm for random groups (random) and clusters obtained on the feature space (group) for samples of the class truck (up) and airplane (down). We see that the optimisation is simpler on the truck class.

5 Conclusion

5.1 Discussion

The question that we tried to solve during this internship is *ill-posed* to a certain extent: How to scientifically interpret the notion of *having a similar impact on the training process*? In a classical Deep Learning setup, a model whose parameters have been randomly initialized learns/is trained by stochastically optimizing on batches of shuffled training data. Re-training a model with a different random seed will typically result in a different model that has a different loss landscape, different parameters, different gradients which will result in a significant predictive variance, even for what we would be tempted to call *ground truth* Leave-one-out influence [**loounfairness**]. This is the reason why the study of the impact of training data on the training process is both useful (any trained model is a trajectory of a random process which have features that can reflect in the model's behavior) and difficult. Insights that we can get about a trained model contains noise that makes results hard to interpret.

During this internship we have studied several approaches to this problem. Iterative methods using influence functions has revealed to be a computationally tractable way to find *functioning groups*. On our 2D toy dataset and model, this method perfectly discriminated between two groups of the same class. However, when applying the same algorithm to the CIFAR-10 dataset, the results are more ambiguous, notably not retrieving near-duplicates of the dataset as members of the same functioning group. In the literature, datamodels don't seems to suffer from this problem. They also allow to control the size of group they're trained on which is not possible with our iterative method. We believe that taking inspiration from this work, as done with the TRAK method, it could be possible to develop a computationally acceptable way to identify functioning groups with enough robustness and interpretability to be considered as a global explainability method.

5.2 Overall internship experience

The DEEL project is a really successful research program that specializes in a few topics and attempt to accelerate progress in every aspect. Scientifically, it fulfilled my desire to delve into the current challenges surrounding trustworthy AI. On a personal level, it presented an opportunity for me to cultivate my software development skills, allowing me to become accustomed to operating within an environment that highly values code contributions in open-source libraries, and where a well-structured DevOps approach is implemented.

One aspect that pleasantly surprised me was the stark contrast between the DEEL team and the academic research teams I had been accustomed to. Instead of the traditional model with a senior professor overseeing a large group of students, the DEEL team comprised managers who skillfully facilitated researchers' work conditions. The team was composed of not only research engineers who actively published papers and developed libraries but also professionals from diverse backgrounds, including detached engineers from prominent companies like Airbus. This dynamic blend of expertise created an environment that felt refreshingly pragmatic and dynamic compared to my previous research internships. This exposure significantly broadened my perspective on how a research team can effectively function and achieve its goals.

Engaging with my specific research project within the DEEL program was both challenging and rewarding. Addressing an exploratory problem with little to no encouraging preliminary work was tough, and at each result that didn't fit my expectations I had to figure out if what was wrong was my implementation or the method itself. This often required me to stretch beyond my comfort zone and grapple with novel problems. While this might have seemed daunting at times, the experience taught me invaluable lessons in perseverance, problem-solving, and adaptability. Each obstacle I encountered became an opportunity for growth, pushing me to develop creative solutions and refine my critical thinking skills. In retrospect, the challenges I faced during the project not only enriched my technical expertise but also raised my confidence in my ability to tackle intricate challenges in the realm of AI and software development. I feel motivated and prepared to start my PhD and that was my ultimate objective. Therefore I consider this internship a success.

A Appendix : additional results on CIFAR-10

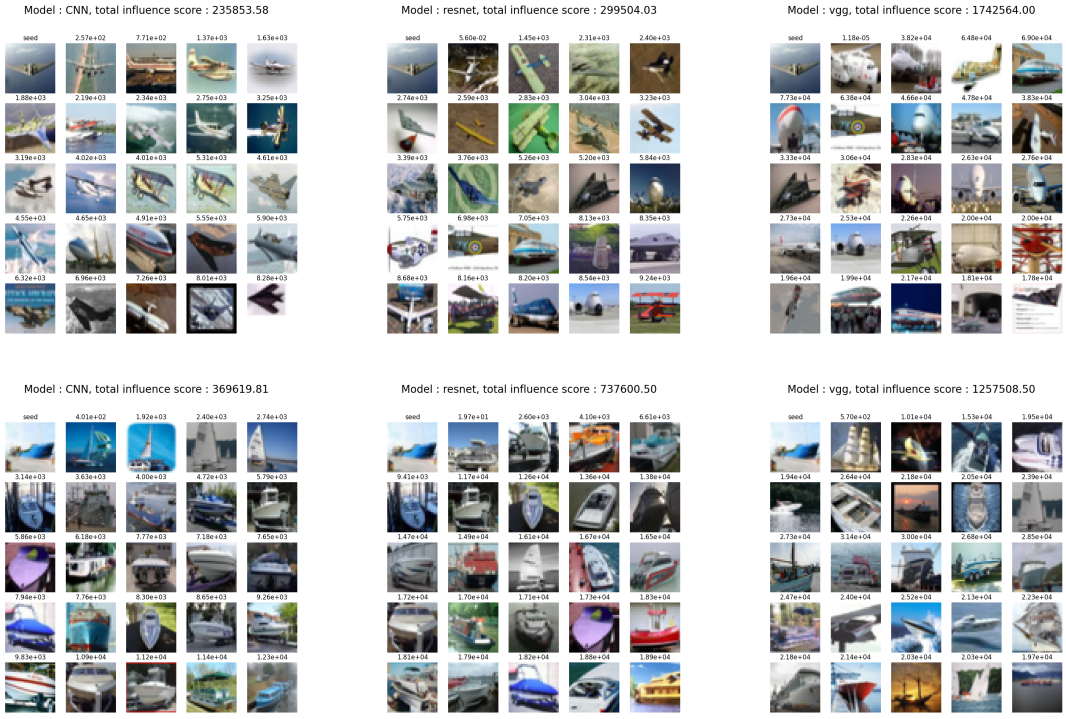


Figure 16 – Additional results similar to 11 with seeds of airplane and ship classes.

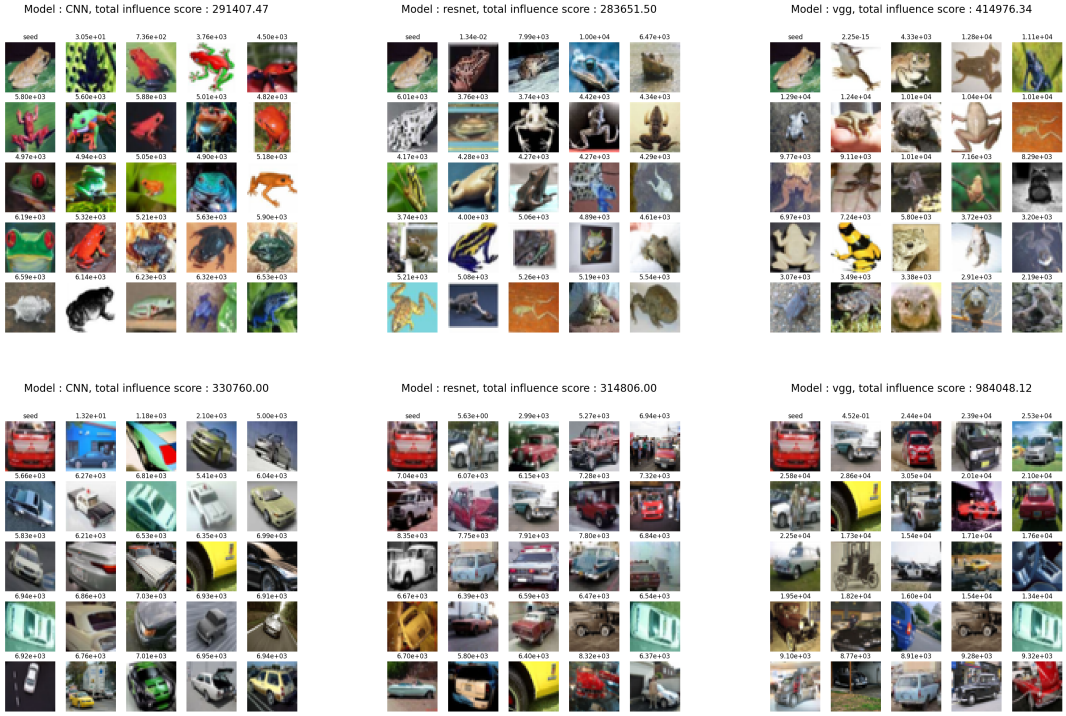


Figure 17 – Additional results similar to 11 with seeds of frog and automobile classes.