Prediction with Uncertainty

Warith HARCHAOUI

When in doubt, tell the truth

Mark Twain *in* More Tramps Abroad, 1897

I respectfully dedicate this chapter to Joannès Vermorel who tried to teach me software programming during summer 2005 in a farm at Fontainebleau within the Centre for Computational Biology headed by Jean-Philippe Vert. Joannès also helped me find a job in summer 2014 although I did not have the right diploma nor skills. Thank you Joannès!

Abstract

Uncertainty estimation with neural networks predictions is not yet well studied (except in the recent and controversial field called Bayesian deep learning). Moreover, in the academic and business worlds, some misinterpretations persist about the probabilities as outputs of a supervised classifier: a probability of belonging to a category or its associated vector across all categories does not fully express information on uncertainty. Indeed, most practitioners extract the maximum entry of a probability vector, which is hacky and not sufficient for a more thorough interpretation of uncertainty. At the same time, in many sensitive applications where security, health or even justice issues are involved, it seems that uncertainty estimation is crucial.

The core idea of our Hypothesis of an Uncertainty Model (HUM) contribution is predicting parameters of an output law instead of predicting an output estimate. On the one hand, putting a parametrized probabilistic law on the output side makes our approach Bayesian. On the other hand, using neural networks to produce these parameters makes it less principled but more pragmatic. By combining the best of these two worlds in a small number of additional lines of code is convenient in practice to estimate uncertainty for automatic or assisted decisions in supervised learning in an engineeringly feasible but yet theoretically principled fashion.

Keywords

Bayesian deep learning, Bayesian neural networks, Uncertainty, Confidence, Reparametrization Trick, Neural Networks, Distributions

Contents

1 Introduction

There is a large class of supervised learning problems that can be cast in an optimization of this form:

$$\min_{\boldsymbol{\theta}_{\mathcal{F}}} \mathcal{L}(\boldsymbol{\theta}_{\mathcal{F}}) \qquad (1)$$
$$\mathcal{L}(\boldsymbol{\theta}_{\mathcal{F}}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}}\left(\ell(\mathbf{y}, \mathcal{F}(\mathbf{x}))\right)$$

where the quantity $\ell(\mathbf{y}, \mathcal{F}(\mathbf{x}))$ expresses how much one tolerates error for confusing the prediction $\mathcal{F}(\mathbf{x})$ with the groundtruth output $\mathbf{y}$ for input $\mathbf{x}$. Our notations abide by the deliberate starting epistemological choices of this dissertation in section **??** especially for the Nature idealized distribution. The prediction function $\mathcal{F}$ is implemented here by a neural network (whose parameters are $\theta_{\mathcal{F}}$) but we insist on the fact that it could be implemented by other tools that we can pick from the enormous machine learning zoo of supervised learning of algorithms [Bishop, 2006] while still keeping this current contribution relevant.

We will describe how to modify this Eq. (1) and how to create the right optimization scheme while still being both mathematically principled and engineeringly friendly in a general case with many applications. But first, we want to debunk some widespread misunderstanding among practitioners in supervised classification about how to interpret an output classification probability vector with respect to uncertainty (which was our engineering starting point). Typically, for an image classification problem (say with an AlexNet convolutional neural network [Krizhevsky et al., 2012] trained on ImageNet [Fei-Fei, 2010]), the output is a 1-sum positive vector of high dimensions (for $K \simeq 10^3$ categories for ImageNet) in the form of conditional probability estimates $(\mathbb{P}(c = k|\mathbf{x}))_{k=1,\ldots,K}$ for an image $\mathbf{x}$ belonging to which category $k \in [\![1, K]\!]$. Unfortunately, there is a detail that is often overlooked concerning to the "knowing part $|\mathbf{x}$" of the notation $\mathbb{P}(c = k|\mathbf{x})$ which states in an implicit (and sometimes forgotten) fashion that the input $\mathbf{x}$ should be taken from the same theoretical distribution both at training and testing times. Indeed, at training time, the formulas are by construction true because the images are taken from Nature (better approximated by large cardinality dataset). At test time, the input examples must be coming from the same Nature distribution for the knowing part of the probability being true which is just a reasonable hypothesis in theory but that is difficult (and rather impossible) to guarantee in practice (i. e. in every real-world and practical industrial applications). The best proof of it is that if we give a completely random input $\mathbf{x}$ taken from an arbitrary distribution, the system would still answer a 1-sum positive vector output although the input does not belong to any of the categories corresponding to these output vector entries due to its independently random nature.

From these considerations, we draw 2 conclusions for each input $\mathbf{x}$ at test time: (i) taking the maximum entry index k^* of such outputs $(\mathbb{P}(c = k|\mathbf{x}))_{k=1,\ldots,K}$ has indeed some significance in a *maximum a posteriori* multinomial perspective in order to make a concrete decision but nevertheless (ii) the corresponding maximum entry estimating $\mathbb{P}(c = k^*|\mathbf{x})$ has no or little uncertainty significance. If uncertainty is a matter of interest, then the output should be considered as a whole K-dimensional point (and not just the maximum of its entries) living on the unit simplex and uncertainty should be a possibly blurry zone around one predicted point in that space. Now that we have described some practical issues related to uncertainty in classification, we present a general solution to tackle supervised learning problems that takes into account some uncertainty information.

The spectacular come back of deep learning techniques can be dated back to 2012, when Krizhevsky et al. [2012] dramatically improved image classification scores. The suprise was considerable because one particular reason (among others): beyond larger experimental scales, there was almost no conceptual difference between the initial LeNet [LeCun and Bengio, 1995] in 1995 and that implementation except: the activation function (from the Sigmoid function to the positive part function) for marginal accuracy improvements, impressive implementation improvements with the original usage of GPU hardware (Graphics Processing Units) for speed (otherwise, training are infeasible for this kind of year-long-type optimization with classic CPUs of traditional computers at that time) and DropOut regularization [Srivastava et al., 2014] to cope with over-parametrization of larger neural networks than usual. Indeed, DropOut consists in injecting artificial random noise to learning paramaters during training with the intuition that without that noise, testing accuracy will be more robustly improved for generalization purposes.

The originality of what Gal [2016] proposed consists in continuing DropOut at test time which implicitly provides (dependent) several outputs for each input and then estimating law parameters on those (say mean and variance of a Gaussian law). Averaging correlated outputs (the learned parameters are very close) has unclear theoretical implications for the Monte Carlo estimation but this method has the merit of pointing to a practical need to investigate sound statistical approaches to cope with the *black-box* deserved reputation of neural networks (outputs without "as-is" possible interpretations).

In the past, Nix and Weigend [1994] proposed to predict the mean and covariance of each prediction by minimizing the Kullback-Leibler divergence between the output empirical data distribution and the Gaussian output prediction (i. e. maximizing the likelihood). It is based on the assumption that there is a sufficiently large data set, i. e. , that their is no risk of overfitting and that the neural network finds the correct regression.

In our work, we chose to suggest a lighter machinery (than the ones of Nix and Weigend [1994] and Gal [2016]): instead of changing learning parameters to implicitly produce an ensemble effect on the prediction side, we directly estimate the parameters of a law on the prediction side. We removed the randomness on the weights that the approach of Gal [2016] implies to put it entirely on the prediction side giving a beneficial *out of local minimum escape* effect to our system while still keeping a valid Kullback-Leibler interpretation like the approach of Nix and Weigend [1994] and even Bishop [1994]. We highly recommend careful readers about uncertainty in deep learning to see the video of Zoubin Ghahramni[1] to get an hindsighful overview of that kind of literature.

As recently describes by Detlefsen et al. [2019], estimating uncertainty is desirable goal in several contexts:

- for time series with Gaussian processes [Seeger, 2004] where means and variances of predictions are estimated at each timestamps;
- the historical work of Bishop [1994] and [Nix and Weigend, 1994] using the seminal idea of predicting the parameters of a law instead of a value that we build upon;
- meanwhile the probabilistic framework of Bayesian techniques was also useful thanks to MacKay [1992] and revisited by Kingma and Welling [2013];
- Monte Carlo Dropout of Gal [2016] with unclear theoretical consequences as predictors are correlated;
- active learning while uncertainty prioritizes choice of informative training examples [Huang et al., 2010].

In this work, we aim to generalize the inspiration of Nix and Weigend [1994] for a larger part of supervised deep learning problems.

2 Revisiting Deep Supervised Learning

As described in the state of the art chapter at section **??**, classification and regression are two important fields applications of supervised learning and are thus both applications of this current uncertainty estimation contribution.

2.1 Classification

Neural Networks are a fundamentally continuous technology that progressively train algorithms by taking into account its own errors on mini-batches of data in order to improve thanks to stochastic gradient descent and back-propagation throughout all the parameters [Goodfellow et al., 2016].

In fact, even plain classification with neural networks is a special case of regression where a degenerated histograms encode categories (a clean category corresponding to a vector with zero entries except for one entry one, hence the name *one-hot encoding*) which has the advantage of naturally manipulating probabilities with float real numbers and of letting us access to the continuous regression tools. For potentially K different categories, the supervised classification literature tends to encourage the negative log-likelihood loss corresponding to:

$$\ell(\mathbf{y}, \mathbf{z}) = -\frac{1}{K}\mathbf{y}^\top \log(\mathbf{z}) = -\frac{1}{K}\sum_{k=1}^{K} \mathbf{y}^{(k)} \times \log(\mathbf{z}^{(k)}) \tag{2}$$

where the label $\mathbf{y}$ adopts the one-hot encoding which gives naturally access to its relaxed probabilistic interpretations.

Depending on the scientific culture of the reader, Eq. (2) can also be seen as the cross-entropy loss or even the Kullback-Leibler divergence between the two discrete densities represented in

[1]`https://www.youtube.com/watch?v=FD8l2vPU5FY`

1-sum positive vectors $\mathbf{z}$ and $\mathbf{y}$. Indeed, the empirical likelihood of given predictions $\mathbf{z} = \mathcal{F}(\mathbf{x})$ measured on (input, output) pairs $(\mathbf{x}, \mathbf{y})$ from training data is

$$\begin{aligned}\text{Likelihood} &= \Big(\prod_{i=1}^{N}\prod_{k=1}^{K}(\mathbf{z}_i^{(k)})^{\mathbf{y}_i^{(k)}}\Big)^{\frac{1}{NK}} \\ &= \Big(\prod_{i=1}^{N}\prod_{k=1}^{K}(\mathcal{F}(\mathbf{x}_i)^{(k)})^{\mathbf{y}_i^{(k)}}\Big)^{\frac{1}{NK}}\end{aligned} \tag{3}$$

and the empirical negative log-likelihood gives:

$$\begin{aligned}-\log(\text{Likelihood}) &= \frac{-1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\mathbf{y}_i^{(k)} \times \log(\mathbf{z}_i^{(k)}) \\ &= \frac{-1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\mathbf{y}_i^{(k)} \times \log(\mathcal{F}(\mathbf{x}_i)^{(k)})\end{aligned} \tag{4}$$

which justifies Eq. (2).

At the same time, if we put the Kullback-Leibler divergence $\text{KL}(\mathbf{y}_i, \mathbf{z}_i)$ definition between two discrete densities $\mathbf{y}_i$ and $\mathbf{z}_i = \mathcal{F}(\mathbf{x}_i)$:

$$\text{KL}(\mathbf{y}_i, \mathbf{z}_i) = \frac{1}{K}\sum_{k=1}^{K}\mathbf{y}_i^{(k)} \times \log\left(\frac{\mathbf{y}_i^{(k)}}{\mathbf{z}_i^{(k)}}\right) = \left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{y}_i^{(k)}\log(\mathbf{y}_i^{(k)})\right) - \log(\text{Likelihood}) \tag{5}$$

but the first term $\frac{1}{K}\sum_{k=1}^{K}\mathbf{y}_i^{(k)}\log(\mathbf{y}_i^{(k)})$ does not depend on the input $\mathbf{x}_i$ nor the prediction $\mathbf{z}_i$ which makes it removable towards a predictor optimization scheme which also justifies the minimization of Eq. (2) in this classification memory aid:

$$\begin{aligned} & \text{Maximizing the classification likelihood} \\ \iff & \text{Minimizing the classification cross-entropy loss} \\ \iff & \text{Minimizing the mean Kullback-Leibler divergence} \\ & \text{between groundtruth labels and predictions discrete densities}\end{aligned} \tag{6}$$

2.2 Regression

For regression, $\mathbf{y}$ is continuous (floats) and possibly multivariate, e. g. the price of rent corresponding to each location in a city, supply chain forecasts, weather forecast maps. A stable and well-studied optimization tool is the quadratic regression:

$$\ell(\mathbf{y}, \mathbf{z}) = \|\mathbf{y} - \mathbf{z}\|_2^2 = \sum_{k=1}^{K}\left(\mathbf{y}^{(k)} - \mathbf{z}^{(k)}\right)^2 \tag{7}$$

another loss is the Manhattan one:

$$\ell(\mathbf{y}, \mathbf{z}) = \|\mathbf{y} - \mathbf{z}\|_1 = \sum_{k=1}^{K}|\mathbf{y}^{(k)} - \mathbf{z}^{(k)}| \tag{8}$$

to focus on small errors (as big errors are considerably more punished in quadratic rather then Manhattan loss).

In a way that is similar to establishing a likelihood for cross-entropy classification, it is possible to draw the links between regression and likelihood. As a matter of fact, the likelihood of a Gaussian model on the output side centered on the predictions $\mathcal{F}(\mathbf{x})$ with an unknown but fixed diagonal homothety covariance matrix $\sigma\mathbf{I}_K$ (for the sake of understanding and lack of any sophisticated hypothesis, it turns out we do not loose generality with $\sigma = 1$ in our particular case):

$$\text{Likelihood} = \left(\prod_{i=1}^{N}\frac{\exp\left(-\frac{1}{2}\|\mathbf{y} - \mathcal{F}(\mathbf{x})\|_2^2\right)}{(2\pi)^{\frac{K}{2}}}\right)^{\frac{1}{N}} \tag{9}$$

as the neg-log-likelihood gets:

$$-\log(\text{Likelihood}) \propto \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \mathcal{F}(\mathbf{x}_i)\|_2^2 \tag{10}$$

and likewise for a the Kullback-Leibler divergence between that Gaussian density q centered at $\mathcal{F}(\mathbf{x})$ (with identity covariance without loss of generality) and the distribution p of ground-truth labels $\mathbf{y} \mid \mathbf{x}$ (when $(\mathbf{x}, \mathbf{y}) \sim$ Nature), we eventually have:

$$\begin{aligned} \text{KL}(p, q) &= \mathbb{E}_{\mathbf{y}\sim q} \log\left(\frac{q(\mathbf{y})}{p(\mathbf{y})}\right) \\ &= \left(\mathbb{E}_{\mathbf{y}\sim q} \log\left(q(\mathbf{y})\right)\right) - \left(\mathbb{E}_{\mathbf{y}\sim q} \log\left(p(\mathbf{y})\right)\right) && (11) \\ & && (12) \end{aligned}$$

but the first term is once again useless for optimizing $\mathcal{F}$, which justifies the least squares approach because:

$$\begin{aligned} -\left(\mathbb{E}_{\mathbf{y}\sim q} \log\left(p(\mathbf{y})\right)\right) &= -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}}\left[\log\left(\frac{\exp\left(-\frac{1}{2}\|\mathbf{y}-\mathcal{F}(\mathbf{x})\|_2^2\right)}{(2\pi)^{\frac{K}{2}}}\right)\right] \\ &\propto \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}}\left(\|\mathbf{y}-\mathcal{F}(\mathbf{x})\|_2^2\right) \end{aligned}$$

In the end we re-establish a regression memory aid:

$$\begin{aligned} & \text{Maximizing the regression likelihood} && (13) \\ \iff & \text{Minimizing the regression quadratic loss} \\ \iff & \text{Minimizing the mean Kullback-Leibler divergence} \\ & \text{between the groundtruth outputs distribution and} \\ & \text{the gaussianized predictions distribution} \end{aligned}$$

Please note that one could do exactly the same for Laplacians and Manhattan L_1 loss instead of Gaussians and quadratic (eucidlean) L_2 criterion.

3 HUM: Hypothesis for an Uncertainty Model

The aim of this work is to propose a way to introduce some uncertainty modelization in supervised machine learning presented above focusing on deep neural networks for the sake of recent research attention. We want to build a model that provides a *probabilistic law* prediction instead of a *value* prediction which is following a Bayesian tradition. At test time, we would use for example the main mode of our output law as the output value (i. e. in lieu of a value prediction) but with some additional uncertainty information depending e. g. on how big the support of the distribution is around that mode.

To put this in simple layman's terms and with some abuse of terminology, traditional approaches try to predict a specific value $\mathcal{F}(\mathbf{x})$ hoping to match the label $\mathbf{y}$ in order to ideally reach even at test time some sort of conformity

$$\mathbf{y} \simeq \mathcal{F}(\mathbf{x}) \tag{14}$$

(approximately) as the loss function evaluation $\ell(\mathbf{y}, \mathcal{F}(\mathbf{x}))$ role is to avoid too much discrepancy during training between prediction $\mathcal{F}(\mathbf{x})$ and groundtruth label $\mathbf{y}$.

In contrast, we propose to focus on uncertainty thanks to outputs corresponding to interpretable parameters of a smooth distribution $\mathcal{G}(\mathbf{x})$ hoping that we reach another kind of conformity

$$\mathbf{y} \approx \mathcal{G}(\mathbf{x}) \tag{15}$$

(approximately). Indeed, the graphic slight difference of notation between the symbols "$\approx$" and "$\sim$" inspired us. More mathematically, this work relies on two statements. First, thanks to the definition of Dirac distributions, we easily establish that we always have:

$$\begin{aligned} (\forall(\mathbf{x},\mathbf{y}) \in \mathbb{R}^D \times \mathbb{R}^K) \;\; \ell(\mathbf{y}, \mathcal{F}(\mathbf{x})) &= \int_{\mathbb{R}^K} \ell(\mathbf{y},\mathbf{z}) \times \delta_{\mathcal{F}(\mathbf{x})}(\mathbf{z}) d\mathbf{z} \\ &\triangleq \mathbb{E}_{\mathbf{z}\sim\delta_{\mathcal{F}(\mathbf{x})}} \left(\ell(\mathbf{y},\mathbf{z})\right) \end{aligned} \tag{16}$$

where $\delta_{\mathbf{a}}$ is the Dirac distribution located at $\mathbf{a}$ that gives that same $\mathbf{a}$ value for when integrated throughout space. Second, as recalled in section **??** with the Robbins-Monro theorem, we do not need to have access at precisely the exact evaluation of the minimization objective or of its gradient in order to minimize it by stochastic gradient descent: only a biased-free estimator of the gradient is required.

The core idea of this chapter is to ask: *Why not taking a more convenient and more interpretable distribution than the Dirac distribution?* in Eq. (16). This intuition is motivated by the fact that using sum Dirac distributions is considering Nature as just a set of scattered points whereas a smooth distribution gives some consistency to our so-called Nature modelization. Now, if we are emancipated from the previous uncertain-free formulation of section 2, then in terms of modelling combining Eq. (16) and Eq. (1) gets:

$$\min_{\mathcal{G}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}} \left[\mathbb{E}_{\mathbf{z}\sim\mathcal{G}(\mathbf{x})} \left[\ell(\mathbf{y},\mathbf{z}) \right] \right] \tag{17}$$

and $\mathcal{G}(\mathbf{x})$ is a prediction law whereas $\mathcal{F}(\mathbf{x})$ previously was a prediction value. A discretized version of this approach has been extensively applied to supply chain forecasts by the Lokad company [Vermorel, 2018] since few years ago. In this chapter, dealing with a wider range of applications (especially suitable with neural networks as we will see) in a more continuous fashion is our contribution to the original and seminal idea of Vermorel [2018].

In terms of optimization of the parameters $\boldsymbol{\theta}_{\mathcal{G}}$ of function $\mathcal{G}$ predicting the output law $\mathcal{G}(\mathbf{x})$, the good news is that the same Robbins-Monro theorem [Robbins and Monro, 1951] as in section **??** justifies the usual Hebbian learning rule of stochastic gradient descent. For the sake of clarity we admit that what is true to the standard stochastic gradient descent algorithms is also true for modern stochastic algorithmical routines like Adam [Kingma and Ba, 2014] which benefits from the rest of regular contemporary deep learning tools [Abadi et al., 2015, Paszke et al., 2017].

For the usual uncertainty-free settings, we usually had:

$$\boldsymbol{\theta}_{\mathcal{F}}^{t+1} = \boldsymbol{\theta}_{\mathcal{F}}^{t} - \alpha^t \hat{\mathbf{f}}^t \tag{18}$$

where α^t is the learning rate and $\hat{\mathbf{f}}^t = \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}_{\mathcal{F}}} \left(\ell(\mathbf{y}_{i_b}, \mathcal{F}(\mathbf{x}_{i_b}))\right)$ with $i_b \sim \mathcal{U}_{\mathbb{N}}(1,N)$ is a random uniform index parsing the N-cardinality training dataset in mini-batches of size B totalling B gradient evaluations per mini-batch.

In our proposed uncertainty modelling, we get:

$$\boldsymbol{\theta}_{\mathcal{G}}^{t+1} = \boldsymbol{\theta}_{\mathcal{G}}^{t} - \alpha^t \hat{\mathbf{g}}^t \tag{19}$$

with same kind of learning rate α^t as before. The main difference is in the bias-free gradient estimation that needs a little Monte Carlo estimation (as M stands for *Monte Carlo number of iterations*):

$$\hat{\mathbf{g}}^t = \frac{1}{MB} \sum_{m=1}^{M} \sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}_{\mathcal{G}}} \left(\ell(\mathbf{y}_{i_b}, \mathbf{z}_{m,i_b})\right) \tag{20}$$

where we still have $i_b \sim \mathcal{U}_{\mathbb{N}}(1,N)$ the same kind of random uniform index as before but we also have M sampled outputs from the same prediction law for each input $\mathbf{x}_{i_b}$ totalling MB gradient evaluations per mini-batch: $\mathbf{z}_{m,i_b} \sim \mathcal{G}(\mathbf{x}_{i_b})$ which is M times more computations than before but there is no guarantee confirming or infirming that a big B or a big M could improve convergence or results as the two sources of stochasticity are related (maybe even a simple $M = B = 1$ scenario could give good resuls).

There is a strong link between our technique and the *Reparametrization Trick* [Kingma and Welling, 2013] for initially variational auto-encoders problematics. Indeed, for the authors, several probabilistic laws, the law parameters can be represented in the gradient calculus from its normalized version (e. g. without mean nor unusual standard deviation for the Gaussian law) of the same simplified probabilistic law accordingly distorted to fit the desired parametrized law. Thanks to three special cases, we will now precisely see how this is all working thanks to pseudo-random generators of our computers.

3.1 Uncertain Logistic Regression for Classification

In logistic regression classification, we recall from section 2.1 that for the general optimization $\min_{\mathcal{F}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}}\left[\ell(\mathbf{y},\mathcal{F}(\mathbf{x}))\right]$ is specified for logistic classification by:

$$\ell(\mathbf{y},\mathbf{z}) = -\frac{1}{K}\mathbf{y}^\top \log(\mathbf{z}) \tag{21}$$

and to maintain the simplex-constraint we propose a uniform law (as segments always remain inside a convex set) for applying Eq. (20):

$$\begin{aligned} & \mathbf{z}_{m,i_b} \sim \mathcal{U}_{\mathbb{R}^K}\left(\mathcal{A}(\mathbf{x}_{i_b}),\mathcal{B}(\mathbf{x}_{i_b})\right) \\ \iff\quad & t_{m,i_b} \sim \mathcal{U}_{\mathbb{R}}(0,1) \text{ and } \mathbf{z}_{m,i_b} = t_{m,i_b}\times\mathcal{A}(\mathbf{x}_{i_b}) + (1-t_{m,i_b})\times\mathcal{B}(\mathbf{x}_{i_b}) \end{aligned} \tag{22}$$

where functions $\mathcal{A}$ and $\mathcal{B}$ give class membership probabilities and are implemented with regular neural networks with a last SoftMax layer and it all becomes:

$$\begin{aligned} \hat{\mathbf{g}}^t &= \frac{1}{MB}\sum_{m=1}^{M}\sum_{b=1}^{B}\nabla_{\boldsymbol{\theta}_{\mathcal{G}}}\left(\ell\left(\mathbf{y}_{i_b}, t_{m,i_b}\times\mathcal{A}(\mathbf{x}_{i_b}) + (1-t_{m,i_b})\times\mathcal{B}(\mathbf{x}_{i_b})\right)\right) \\ &= \frac{1}{MB}\sum_{m=1}^{M}\sum_{b=1}^{B}\nabla_{\boldsymbol{\theta}_{\mathcal{G}}}\left(-\mathbf{y}_{i_b}^\top \log\left(t_{m,i_b}\times\mathcal{A}(\mathbf{x}_{i_b}) + (1-t_{m,i_b})\times\mathcal{B}(\mathbf{x}_{i_b})\right)\right) \end{aligned} \tag{23}$$

to be compared to Eq. (20).

In these settings, the prediction uncertainty can be measured in $0 \le \frac{1}{2}\|\mathcal{A}(\mathbf{x}_{i_b}) - \mathcal{B}(\mathbf{x}_{i_b})\|_1 \le 1$ which justifies our probabilistic model. In terms of Kullback-Leibler divergence, minimizing the mean divergence between the fixed discrete groundtruth density $\mathbf{y}$ and the random discrete prediction density $\mathbf{z} \sim \mathcal{G}(\mathbf{x})$ this way, is useful even at a category level by superposing both $\mathcal{A}(\mathbf{x})$ and $\mathcal{B}(\mathbf{x})$ histograms (i. e. positive 1-sum vectors).

Out of curiosity, we can explore what is happening for more than 2 histograms for the sake of algorithms.

$$\begin{aligned} & \mathbf{z}_{m,i_b} \sim \mathcal{U}_{\mathbb{R}^K}\left(\mathcal{A}_1(\mathbf{x}_{i_b}),\dots,\mathcal{A}_c(\mathbf{x}_{i_b}),\dots,\mathcal{A}_C(\mathbf{x}_{i_b})\right) \\ \iff\quad & \mathbf{t}_{m,i_b} \sim \boldsymbol{\Delta}_C \text{ and } \mathbf{z}_{m,i_b} = \sum_{c=1}^{C}\mathbf{t}_{m,i_b}^{(c)}\times\mathcal{A}_c(\mathbf{x}_{i_b}) \end{aligned} \tag{24}$$

where $\boldsymbol{\Delta}_C$ is the uniform distribution over the C-order simplex (i. e. $\sum_{c=1}^{C}\mathbf{t}_{m,i_b}^{(c)} = 1$). To uniformly sample inside that C-order simplex, first, we first sample uniformly $C-1$ values between 0 and 1 $u_{m,i_b,c} \sim \mathcal{U}_{\mathbb{R}}(0,1)$ (and we rearranged the indices corresponding to c such that they are re-ordered in increasing order), second we recursively build:

$$\begin{aligned} \mathbf{t}_{m,i_b}^{(1)} &= u_{m,i_b,1} \\ \mathbf{t}_{m,i_b}^{(c)} &= u_{m,i_b,c+1} - u_{m,i_b,c} \text{ for } c \in [\![2, C-1]\!] \\ \mathbf{t}_{m,i_b}^{(C)} &= 1 - u_{m,i_b,C-1} \end{aligned} \tag{25}$$

that cancels out in a sum of 1 and finally we get:

$$\hat{\mathbf{g}}^t = \frac{1}{MB}\sum_{m=1}^{M}\sum_{b=1}^{B}\nabla_{\boldsymbol{\theta}_{\mathcal{G}}}\left(\ell\left(\mathbf{y}_{i_b}, \sum_{c=1}^{C}\mathbf{t}_{m,i_b}^{(c)}\times\mathcal{A}_c(\mathbf{x}_{i_b})\right)\right) \tag{26}$$

which implies an output law that fires in a convex set drawn by the polygon made of $(\mathcal{A}_c(\mathbf{x}))_{c=1,\dots,C}$ for each input $\mathbf{x}$. Although this formulation is theoretically pleasing, it did not provide, in our experience any additional interpretation ease nor accuracy improvements, yet.

3.2 Uncertain Least Square for Regression

Likewise, in least square regression:

$$\ell(\mathbf{y},\mathbf{z}) = \|\mathbf{y}-\mathbf{z}\|_2^2 \tag{27}$$

and we choose the Gaussian distribution to obtain:

$$\begin{aligned} & \mathbf{z}_{m,i_b} \sim \mathcal{N}\left(\boldsymbol{\mu}(\mathbf{x}_{i_b}), \boldsymbol{C}(\mathbf{x}_{i_b}) \times \boldsymbol{C}(\mathbf{x}_{i_b})^\top\right) \\ \Longleftrightarrow \quad & \boldsymbol{\epsilon}_{m,i_b} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_K) \text{ and } \mathbf{z}_{m,i_b} = \boldsymbol{\mu}(\mathbf{x}_{i_b}) + \boldsymbol{C}(\mathbf{x}_{i_b}) \times \boldsymbol{\epsilon}_{m,i_b} \end{aligned} \tag{28}$$

where function $\boldsymbol{\mu}$ is a vanilla neural network and $\boldsymbol{C}$ is a lower-triangular matrix (with possible bounded diagonal values thanks to affine Sigmoids see Table **??** at page **??**). For the Hebbian rule we update with:

$$\hat{\mathbf{g}}^t = \frac{1}{BM} \sum_{b=1}^{B} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}_{\mathcal{G}}} \left(\|\mathbf{y}_i - (\boldsymbol{\mu}(\mathbf{x}_{i_b}) + \boldsymbol{C}(\mathbf{x}_{i_b}) \times \boldsymbol{\epsilon}_{m,i_b})\|_2^2 \right) \tag{29}$$

In these settings, the uncertainty information is contained in the $\boldsymbol{C}(\mathbf{x}_{i_b}) \times \boldsymbol{C}(\mathbf{x}_{i_b})^\top$ covariance matrix as we have fitted a Gaussian to our prediction outputs. Obviously, for the sake of completeness, we have modelled a Gaussian distribution with a full covariance matrix but a diagonal covariance matrix or a simple proportional to the identity covariance matrix is most of the times enough in real-world applications.

In terms of Kullback-Leibler divergence, we still have the same probabilistic interpretations as we minimize the mean over input $\mathbf{x}$ of the divergence between the deterministic groundtruth labels $\mathbf{y}$ Dirac distribution and the smooth Gaussian distribution $\mathcal{N}\left(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{C}(\mathbf{x}) \times \boldsymbol{C}(\mathbf{x})^\top\right)$ which means we did manage not to sacrifice interpretation for uncertainty. The mean $\boldsymbol{\mu}(\mathbf{x})$ plays the role of the good old-fashioned value prediction that is tainted by incertainty $\boldsymbol{C}(\mathbf{x}) \times \boldsymbol{C}(\mathbf{x})^\top$.

3.3 Uncertain Mixtures for Regression and Classification

Sometimes, beyond mono-modal distribution for regression (e. g. often Gaussian) and uniform law for classification, one can imagine a non-continuous spectrum of outcomes like in supply chain forecasts [Vermorel, 2018] to handle several discrete scenarios at the same time. This way, one can benefit from the computational power of machines to process newly predictable outcomes. This is the reason why for each input $\mathbf{x}$ we tackle here one finite mixture $q(\mathbf{x})$ of C distributions with proportions $\boldsymbol{\pi}(\mathbf{x})$ and components $(g_k(\mathbf{x}))_{k=1}^{K}$ (that are themselves distributions) which can written as:

$$q(\mathbf{x}) = \sum_{c=1}^{C} \boldsymbol{\pi}_c(\mathbf{x}) \times g_c(\mathbf{x}) \tag{30}$$

Just by the continuous integral definition of the expectation, we obtain:

$$\mathbb{E}_{\mathbf{z}\sim q}\left(\ell(\mathbf{y}, \mathbf{z})\right) = \sum_{c=1}^{C} \boldsymbol{\pi}_c(\mathbf{x}) \times \mathbb{E}_{\mathbf{z}_c \sim g_c(\mathbf{x})}\left[\ell(\mathbf{y}, \mathbf{z}_c)\right] \tag{31}$$

which facilitates our mixture use because we can directly use the formulas of the previous sections 3.1 and 3.2 depending on the applications we have:

$$\hat{\mathbf{g}}^t = \frac{1}{MB} \sum_{m=1}^{M} \sum_{b=1}^{B} \sum_{c=1}^{C} \nabla_{\boldsymbol{\theta}_{\mathcal{G}}} \left(\boldsymbol{\pi}_c(\mathbf{x}) \times \ell(\mathbf{y}_{i_b}, \mathbf{z}_{m,i_b,c})\right) \text{ with } \mathbf{z}_{m,i_b,c} \sim \mathcal{G}_c(\mathbf{x}_{i_b}) \tag{32}$$

where the previous technique of reparametrization as previously can re-applied to generate samples from $\mathcal{G}_c(\mathbf{x}_{i_b})$.

In this mixture scenario, the parameters set $\boldsymbol{\theta}_{\mathcal{G}}$ contains the parameters $\theta_{\boldsymbol{\pi}}$ of the neural network $\boldsymbol{\pi}$ that gives the C proportions and each of the parameters $\boldsymbol{\theta}_{\mathcal{G}_c}$s that parametrize the laws $\mathcal{G}_c(\mathbf{x})$s. Two special cases of mixtures attract our attention:

Gaussian Mixture for regression Interestingly, this corresponds to the Mixture Density Networks designed by Bishop [1994] in the 1990s;

Dirac Mixture for classification For a high number of possible categories ($K \simeq 10^3$ in ImageNet [Fei-Fei, 2010]), allowing a mixture of a much reduced number $C \sim 10^1$ of components is useful to handle both uncertainty and close categories in a more interpretable way than in the usual supervised deep learning apparatus.

e

3.4 Links with Other techniques

Interestingly, our straightforward yet efficient approach has some connections with other techniques. Let us take into account that the discrepancy function ℓ in Eq. (17), if we recall that in classification and regression cases, we chose $\ell(\mathbf{y}, \mathbf{z}) = -\mathbf{y}^\top \log(\mathbf{z})$ and $\ell(\mathbf{y}, \mathbf{z}) = \|\mathbf{y} - \mathbf{z}\|_2^2$ respectively. In both cases, for any label $\mathbf{y}$, we know that the function $\mathbf{z} \mapsto \ell(\mathbf{y}, \mathbf{z})$ is convex and thus, thanks to Jensen inequality for any distribution q of predicted output (e. g. $q = \mathcal{G}(\mathbf{x})$):

$$\mathbb{E}_{\mathbf{z}\sim q}\left(\ell(\mathbf{y}, \mathbf{z})\right) \geq \ell\left(\mathbf{y}, \mathbb{E}_{\mathbf{z}\sim q}(\mathbf{z})\right) \tag{33}$$

which implies that:

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}}\left[\mathbb{E}_{\mathbf{z}\sim\mathcal{G}(\mathbf{x})}\left[\ell\big(\mathbf{y}, \mathbf{z}\big)\right]\right] \geq \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\text{Nature}}\left[\ell\Big(\mathbf{y}, \mathbb{E}_{\mathbf{z}\sim\mathcal{G}(\mathbf{x})}\big(\mathbf{z}\big)\Big)\right] \tag{34}$$

which means that for a non-Dirac global minimum $\mathcal{G}^*$ of Eq. (17) corresponding to the lower bounded right hand side of Eq. (34), we know that the global minimum is also reached at

$$\mathcal{G}_d : \mathbf{z} \mapsto \mathbb{E}_{\mathbf{z}\sim\mathcal{G}^*(\mathbf{x})}(\mathbf{z}) \tag{35}$$

One possible interpretation of that could be *The deterministic optimization and our optimization share some global minima if they exist*. What is interesting to mention is that in practice, first, space zones that is very close to the training data manifold have roughly the same result for with or without uncertainty model and second, *in-between* zoom have better results and interpretation with out uncertainty model. Another way to look at it is to consider our technique like a DropOut [Srivastava et al., 2014] technique consisting in adding some randomness to avoid being stuck in local minima. Our approach also looks like the one of **?** but we do linear combination on the output instead of doing it at the input like them. Our method needs further investigation beyond showing good results because of intringuing self-regularized optimization phenomena thanks to randomness. Although, it is rigorous to separate model from optimization, in our case, it is possible that the additional randomness of our optimization technique actual makes our model more robust and it is unclear to attribute improvements between model and optimization as we will see in the next section.

4 Algorithm and Practical Implementation Details

Based on the updated Hebbian rules, we previously described in normal and out uncertain contexts, the stochastic gradient descent algorithm optimizes the parameters of our neural networks. Using the (pseudo)-random number generators of our computers, we were able to generalize the Robbins-Monro theorem allowing us to calculate our bias-free estimators of gradients at each stochastic gradient iteration.

4.1 Safe Computations

Some numerical overflow instabilities have been studied and avoided thanks to the so-called *logsumexp* trick. This technique must be revisited in our case. Let's first briefly describe what it usually is and then how we can adapt it for our implementations.

Even though the *logsumexp* trick can be used in many scenarios, we choose to describe it in the special example of the previouly presented logistic regression of classification for the sake of pragmatism. In neural networks usual implementations, the cross-entropy loss is applied after the computation of a SoftMax layer on what we call logits $\mathbf{c} = [\mathbf{c}_1, \ldots, \mathbf{c}_k, \ldots, \mathbf{c}_K]$ (see the previously described in Table **??** at page **??**). At some point, there is a need to compute the gradient of the quantity $\mathbf{v} \in \mathbb{R}^K$:

$$\mathbf{v} = \log(\text{SoftMax}(\mathbf{c})) \tag{36}$$

with respect to the neural network parameters involved in the calculus of $\mathbf{c}$. If we take a closer look to $\mathbf{v}$, we expand it to:

$$\mathbf{v}_k = \log\left(\frac{\exp(\mathbf{c}_k)}{\sum_{k'=1}^{K}\exp(\mathbf{c}_{k'})}\right) = \log(\exp(\mathbf{c}_k)) - \log\left(\sum_{k'=1}^{K}\exp(\mathbf{c}_{k'})\right) \tag{37}$$

The direct computation of $\exp(\mathbf{c}_k)$ where $\mathbf{c}_k \geq 30$ is already bigger than 10^{13} which makes it all unfeasible for a modern computer even though we know that $\log(\exp(\mathbf{c}_k)) \simeq \log(10^{13}) \simeq 30$ is a reasonable number. For any value $M \in \mathbb{R}$, we can use the fact that:

$$\begin{aligned} \mathbf{v}_k &= M + \log(\exp(\mathbf{c}_k - M)) - M - \log\left(\sum_{k'=1}^{K} \exp(\mathbf{c}_{k'} - M)\right) \\ &= \log(\exp(\mathbf{c}_k - M)) - \log\left(\sum_{k'=1}^{K} \exp(\mathbf{c}_{k'} - M)\right) \end{aligned} \tag{38}$$

and for the specific choice $M = \max_k \mathbf{c}_k$, our numerical problems disappear because all exponential arguments $\mathbf{c}_k - M \leq 0$ are more suitable for the computation of $\exp(\mathbf{c}_k - M) \leq 1$ without approximation.

Nowadays deep learning tools such as PyTorch [Paszke et al., 2017], Tensorflow [Abadi et al., 2015] and MXNet [Chen et al., 2015] applies that *logsumexp* trick implicitly *behind the scene* but this can reused in this present work. We can remark that the idea consisted in avoiding big exponentials by guaranteeing negativity of their arguments thanks to the maximum entry that is substracted for safer computations.

Likewise, in our uncertainty estimation context, we have to compute at some point the quantity $\mathbf{w} \in \mathbb{R}^K$ for $t \in [0, 1]$:

$$\mathbf{w} = \log\left(t \times \text{SoftMax}(\mathbf{a}) + (1-t) \times \text{SoftMax}(\mathbf{b})\right) \tag{39}$$

where $\mathbf{a} \in \mathbb{R}^K$ and $\mathbf{b} \in \mathbb{R}^K$ come from neural networks. With that negative argument exponentials technique in mind, we understand that the SoftMax function is invariant to uniform shifting:

$$(\forall M \in \mathbb{R}) \;\; \text{SoftMax}(\mathbf{c}) = \text{SoftMax}(\mathbf{c} - M) \tag{40}$$

thus we propose for any $P \in \mathbb{R}$ and $Q \in \mathbb{R}$:

$$\begin{aligned} \mathbf{w}_k &= \log\left(t \times \frac{\exp(\mathbf{a}_k)}{\sum_{k'=1}^{K} \exp(\mathbf{a}_{k'})} + (1-t) \times \frac{\exp(\mathbf{b}_k)}{\sum_{k'=1}^{K} \exp(\mathbf{b}_{k'})}\right) \\ &= \log\left(t \times \frac{\exp(\mathbf{a}_k - P)}{\sum_{k'=1}^{K} \exp(\mathbf{a}_{k'} - P)} + (1-t) \times \frac{\exp(\mathbf{b}_k - Q)}{\sum_{k'=1}^{K} \exp(\mathbf{b}_{k'} - Q)}\right) \end{aligned}$$

and to guarantee the same negative exponentials properties as before, we choose: $P = \max_k \mathbf{a}_k$ and $Q = \max_k \mathbf{b}_k$ which provides much more numerical stability when it comes to computing $\mathbf{w}$ and its gradient with respect to the parameters *a fortiori*.

In the special case where the number of classes $K = 2$, practioners use a Sigmoid function applied on a logit value c and the 2 estimated probabilities get $[\text{Sigmoid}(c), 1 - \text{Sigmoid}(c)]$ but the SoftMax function is a generalization of the Sigmoid for that $K = 2$ binary case thanks to that relationship:

$$\begin{aligned} \text{SoftMax}\left([0, -c]\right) &= [\text{Sigmoid}(c), \text{Sigmoid}(-c)] \\ &= [\text{Sigmoid}(c), 1 - \text{Sigmoid}(c)] \end{aligned} \tag{41}$$

which gives us Eq. (41) in that $K = 2$ binary special case for no supplementary effort.

4.2 Re-Using Uncertainty-Free Models

In 2018, Apple and Amazon reached $\$10^{12}$ of market capitalization and in 2019 Microsoft followed in 2020 by Alphabet that owns Google also did reach that financial milestone while Facebook is at roughly $\$0.6 \times 10^{12}$. So we humbly imagine a way to use all these powerful open-source means to better train our uncertainty models just to *stand on the shoulders of giants*.

In order to re-use all that wealth of trained models, we use the idea that an uncertainty-free model is a model that has no uncertainty: for example, a Gaussian density with no covariance becomes a Dirac distribution, a uniform density with equal bounds also becomes a Dirac distribution. We can briefly explain this strategy in two supervised instances:

Regression In our Gaussian regression case, $\boldsymbol{\mu}$ can be reasonably well-initialized by its uncertainty-free already trained counterpart with a zero-initialized Cholesky function $\mathbf{C}$ for the covariance matrix.

$$\mathbf{C} = \mathbf{0} \iff \text{no uncertainty} \tag{42}$$

Classification In our uniform classification case, both $\mathcal{A}$ and $\mathcal{B}$ can be initialized by the same weights (while the random generator of t will give different gradients starting from the very first mini-batch gradient computation).

$$\mathcal{A} = \mathcal{B} \iff \text{no uncertainty} \tag{43}$$

In our experiments, approximately 10% more of training epochs is enough to get relevant additional uncertainty information. We must insist on the fact that we did not want to improve accuracy of already impressively sophisticated algorithms (at the very least with an engineering perspective and for means of distributed computational power) but only we only wanted to get some new uncertainty information.

4.3 Algorithms

There exists many ways to implement the neural networks from our HUM (Hypothesis of Uncertainty Model). In this section, we provide a binary classification pseudo-code and another pseudo-code for regression.

Neural Networks are often built in layers which correspond to a composition of elementary functions (even in deep residual learning [He et al., 2016], we can still separate blocks of layers). This is good news for cheap learning of heavy neural networks with uncertainty because they can be initialized with worldwide companies means but without uncertainty. For example, in classification, a neural network corresponding to a function $\mathcal{F}$ can be thus separated it into $\mathcal{F} = \mathcal{G} \circ \mathcal{H}$ and we create 2 functions, namely $\mathcal{A} = \mathcal{G}_\mathcal{A} \circ \mathcal{H}$ and $\mathcal{B} = \mathcal{G}_\mathcal{B} \circ \mathcal{H}$ where function $\mathcal{H}$ is shared whereas both $\mathcal{G}_\mathcal{A}$ and $\mathcal{G}_\mathcal{B}$ are equally initialized by the previous training output $\mathcal{G}$.

Algorithm 1 Classification with Uncertainty

```
1: Input: Data $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\dots,N}$ where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{y}_i \in \{0,1\}^K \cap \Delta_K$, a mini-batch size $B$ and a Monte Carlo parameter $M \in \mathbb{N}^*$
2: Initialization: $\theta = \{\theta_\mathcal{H}, \theta_{\mathcal{G}_\mathcal{A}}, \theta_{\mathcal{G}_\mathcal{B}}\}$ initialized from a regular uncertainty-free classification optimization scheme for respectively $\mathcal{H}, \mathcal{A}$ and $\mathcal{B}$;
3: while $\theta$ has not converged do
4:     Free all gradients accumulators
5:     Sample a mini-batch of size $B$ from the dataset
           $$(\mathbf{x}_{i_b}, \mathbf{y}_{i_b})_{b=1,\dots,B} \text{ where } i_b \sim \mathcal{U}_\mathbb{N}(1, N)$$
6:     Sample $BM$ values from a pseudo-random generator
           $$t_{b,m} \sim \mathcal{U}_\mathbb{R}(0,1) \quad \text{for } b = 1, \dots, B \text{ and } m = 1, \dots, M$$
7:     Compute
           $$\mathbf{h}_b \leftarrow \mathcal{H}(\mathbf{x}_{i_b}),\ \mathbf{a}_b \leftarrow \mathcal{G}_\mathcal{A}(\mathbf{h}_b) \text{ and } \mathbf{b}_b \leftarrow \mathcal{G}_\mathcal{B}(\mathbf{h}_b) \quad \text{for } b = 1, \dots, B$$
8:     Apply the logsumexp trick $\mathbf{a}_b \leftarrow \mathbf{a}_b - \max_k \mathbf{a}_b^{(k)}$ and $\mathbf{b}_b \leftarrow \mathbf{b}_b - \max_k \mathbf{b}_b^{(k)}$
9:     And finally
           $$\mathcal{L} \leftarrow \frac{-1}{BM} \sum_{m=1}^{M} \sum_{b=1}^{B} \mathbf{y}_{i_b}^\top \log\left(t_{b,m} \times \mathrm{SoftMax}(\mathbf{a}_b) + (1 - t_{b,m}) \times \mathrm{SoftMax}(\mathbf{b}_b)\right)$$
10:    Perform a gradient descent step of $\mathcal{L}$ to update $\theta$
11: end while
```

Algorithm 2 Regression with Uncertainty

```
1: Input: Data $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\dots,N}$ where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{y}_i \in \mathbb{R}^K$, a mini-batch size $B$ and a Monte Carlo
   parameter $M \in \mathbb{N}^*$
2: Initialization: $\theta_\mu$ for a mean function $\boldsymbol{\mu}$ initialized from a regular uncertainty-free regression
   optimization scheme; $\theta_\mathbf{C}$ for a lower triangular matrix function $\mathbf{C}$ initialized by a diagonal of
   training standard deviations; $\theta = \{\theta_\mu, \theta_\mathbf{C}\}$
3: while $\theta$ has not converged do
4:     Free all gradients accumulators
5:     Sample a mini-batch of size $B$ from the dataset
           $(\mathbf{x}_{i_b}, \mathbf{y}_{i_b})_{b=1,\dots,B}$ where $i_b \sim \mathcal{U}_\mathbb{N}(1, N)$
6:     Sample $BM$ values from a pseudo-random generator
           $\boldsymbol{\epsilon}_{b,m} \sim \mathcal{N}(\mathbf{0}_K, \mathbf{I}_K)$
7:     Compute
           $\mathbf{m}_b \leftarrow \boldsymbol{\mu}(\mathbf{x}_{i_b})$ and $\mathbf{S}_b \leftarrow \mathbf{C}(\mathbf{x}_{i_b})$
8:     The loss gets:
           $\mathcal{L} \leftarrow \frac{1}{BM} \sum_{m=1}^{M} \sum_{b=1}^{B} \|\mathbf{m}_b + \mathbf{S}_b \times \boldsymbol{\epsilon}_{b,m} - \mathbf{y}_{i_b}\|_2^2$
9:     Perform a gradient descent step of $\mathcal{L}$ to update $\theta$
10: end while
```

5 Experiments

5.1 Synthetic data

In order to go deeper with our approach, we study a toy dataset called "Two Moons" with 1000 points for each of the 2 groups (blue and red 2D dots) in 2 dimensions as presented in Fig. 1 in two scenarios: an easily separable one and another much noisier one. In this supervised classification

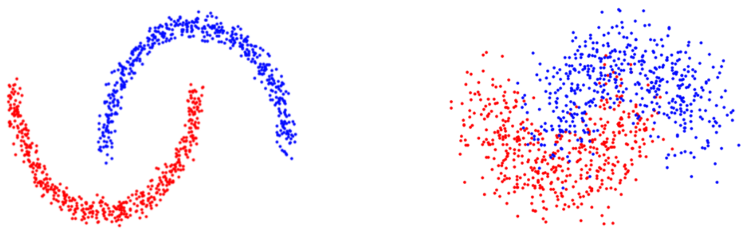
Figure 1: "Two Moons" Dataset in two Scenarios: a clear and easy case (left) and a noisy and difficult case (right)

exercize, the easy case wihtout noise in not very interesting from an uncertainty perspective which makes our approach unnecessary on the left. On the contrary, the difficult case is more relevant because some space regions imply legitimate doubt levels on the right.

We trained a small multi-layer perceptron on this dataset to investigate what was empirically at stake: for each input $[x_1, x_2]^\top$ we predict 2 probability vectors $p = [a, b]^\top$ and $q = [c, d]^\top$ as bounds of a uniform prediction law $\hat{\mathbf{y}} \sim \mathcal{U}_\mathbb{R}(p, q)$ following section 3.1 (page 8). Thus, it is fair to consider $u = \frac{\|p-q\|}{2} \in [0, 1]$ as an evaluation of uncertainty. Our goal is not to provide any classification accuracy improvements but only some more additional information about uncertainty. So we empirically checked on this synthetic problem if our technique was able to preserve good classification results which is the case in Fig. 2 with yellow for the *blue class*, a purple for the *red class* and small region of green for the blurry boundary with a thickness that is understandable when considering the mixed classes. Now we can see if it is also providing a useful uncertainty information in Fig. 3.

Here we see an interesting phenomenon: the prediction is highly confident almost everywhere in space (purple low level of uncertainty) except in the boundary which can be split in two: (i) low uncertainty boundary in purple at the center of the figure and high uncertainty far from where the points are (whereas the boundary is green in the previous Fig. 2 wihtout distinction of having a 0.5 probability of being blue.)

These encouraging synthetic results exhibits two ways of *not knowing*: with uncertainty in yellow in Fig. 3 and with certainty in purple in Fig. 3. The Dirac phenomenon we suspected in section 3.4 is occuring: close to the training manifold, the uniform distribution we get is a Dirac but going further from the training manifold grows some thickness away from the Dirac distribution.

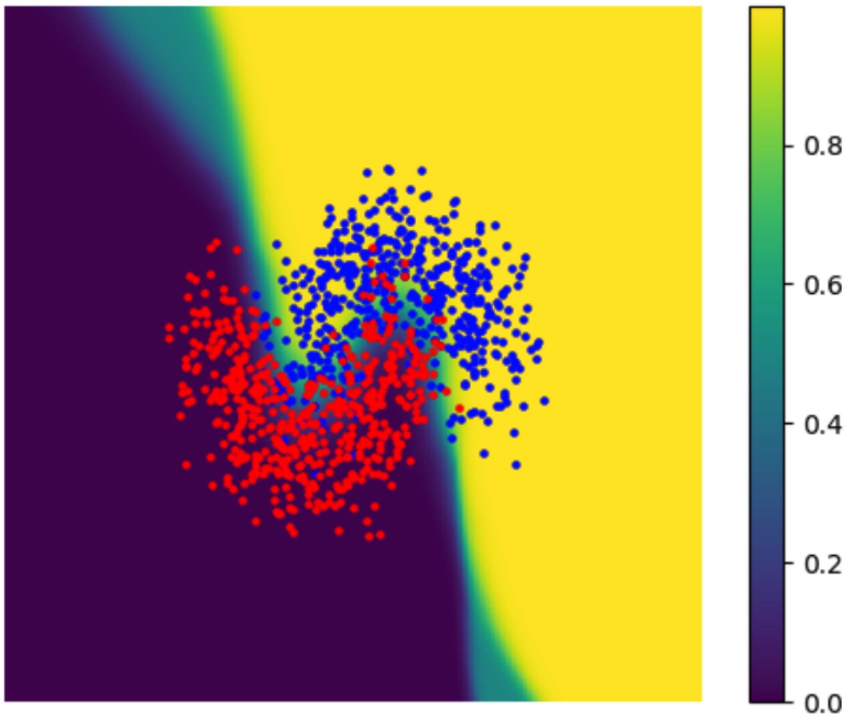

Figure 2: "Two Moons" Map of Prediction Probability of *being blue* $\frac{a+c}{2}$

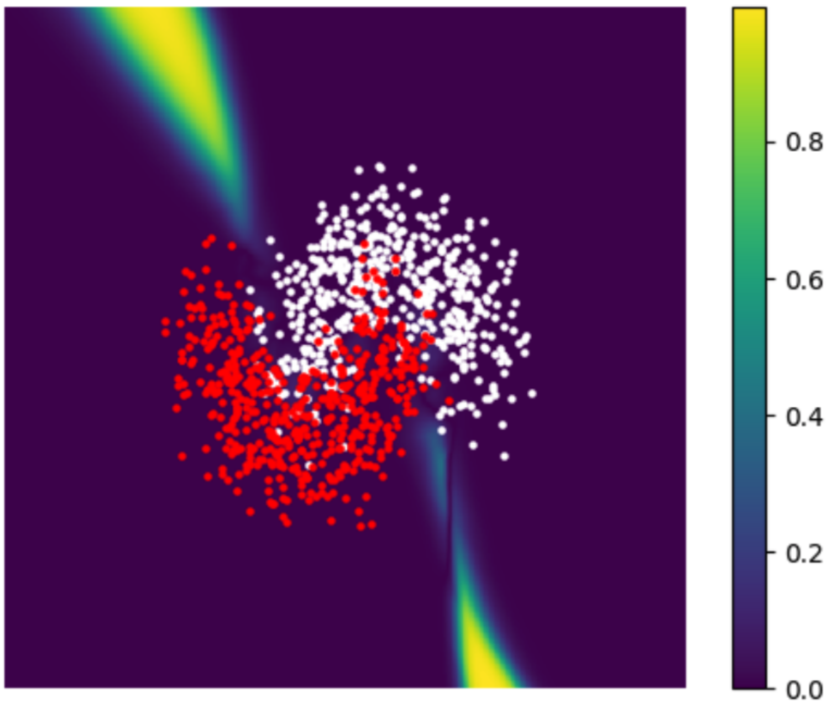

Figure 3: "Two Moons" Map of Uncertainty Probability $\frac{\|p-q\|_1}{2}$ (the higher, the less certain). Blue became white for readibility reasons.

5.2 Dogs and Wolves

To illustrate our approach, we took a image classification task that is difficult even for us as human beings: distinguishing domesticated canide animals (say mainly dogs) and wild canide animals (say wolves, foxes etc.) that we took from the ImageNet dataset [Fei-Fei, 2010] (and its tree-structured label ontology). We formed a dataset[2] of a balanced 800 images per class training set and 200 images per class validation set on which we measure accuracy. We also formed a test set from the ImageNet test dataset (annotated by the ResNet-152 available via PyTorch) with 100 cats, 100 dogs and 100 wolves to evaluate test error and extrapolation uncertainty (i. e. out of distribution) as cats are not wolves nor dogs [3].

(a) Dog (b) Wolves

Figure 4: Two examples of images from our "Dogs and Wolves" dataset taken from Imagenet

As Greenspan et al. [2016] explains, using a convolution neural networks (CNN) trained from a *natural* images classification problem with plenty of labeled data (e. g. ImageNet [Fei-Fei, 2010]) is useful for a *medical* images classification problem that suffered from a notorious lack of large scale labeled datasets until recently. Indeed, *chopping the head off* a CNN is relevant as a feature extractor even for a very different problem which is understandable because early computer vision cues such as the relationship between contours and gradients in the first layers seems to be independent from the image domain and thus shared among all computer vision tasks. Meanwhile, some empirical studies tend to show that the first layers of a convolutional network often correspond to Gabor functions for general-purpose computer vision applications and the deeper layers can be interpreted in more sophisticated ways [Zeiler and Fergus, 2014].

Inspired by these successful ideas and intuitions, first we build a convolutional neural network like in Fig. 5 close to AlexNet [Krizhevsky et al., 2012] but we double the number of ending layers and second we make all the first layers shared for two separate ending dense neural networks. Indeed for our "Dogs and Wolves" images classification problem, we need two outputs with the belief that some features are shared among these virtually two neural networks. We do not entirely double the number of parameters with two separate convolutional neural networks because we want to prevent our learning procedure from some bad overfitting effects of over-parametrization and gain some more speed.

Our "Dogs and Wolves" images classification problem looks like our "Two Moons" noisy case with much higher dimensionality and where even human beings might disagree sometimes. In fact, the very notion of being domesticated is human-related for example. Numerically, we observe a slight improvement of less than 1% of validation accuracy from a 76% in normal uncertainty-free training and 77% with our HUM technique. In fact, we initialized as previously stated in section 4.2 our HUM convolutional neural network with a classic uncertainty-free regular approach. The goal here was not to improve the validation score but only to check if we are not loosing some which is valid as Fig. 6 is showing little overlaid differences. The real benefit of our HUM approach is the uncertainty estimation. Thus, it seems that our HUM approach on a difficult computer vision task is efficient.

In Table. 1 we provide some of the most uncertain examples according to our HUM CNN: we took the top-8 uncertain (considering $\frac{\|p-q\|_1}{2}$ as an evaluation of uncertainty) images of our testing

[2]`http://harchaoui.org/warith/dogs_wolves.zip`
[3]`http://harchaoui.org/warith/imagenet_test_resnet152_pytorch.zip`

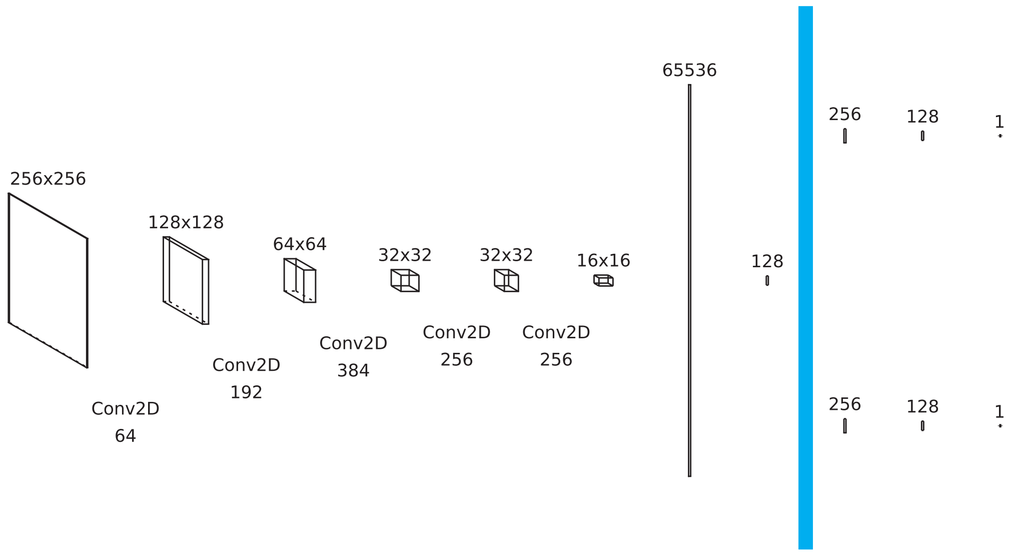

Figure 5: Our Convolutional Neural Network in two parts separated by the tick blue line: (i) some shared convolutional and fully-connected layers on the left and (ii) a few fully-connected layers on the right to finally output the two bounds of our uniform prediction law.

set (that includes cats that were not seen during training). Qualitatively, we see that just taking the entropy of a regular CNN as an uncertainty criterion does not perform as well in Table. 2.

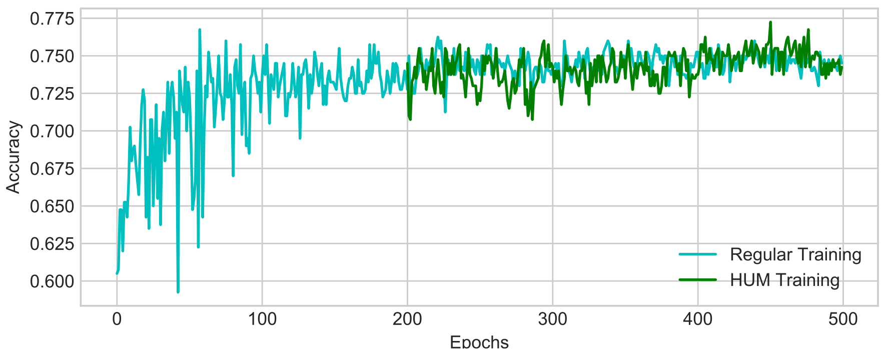

Figure 6: Validation score w.r.t. the training epochs comparing our HUM CNN with its initialized regular CNN counterpart (the higher, the better, 1.0 means 100% of good classification on the validation set)

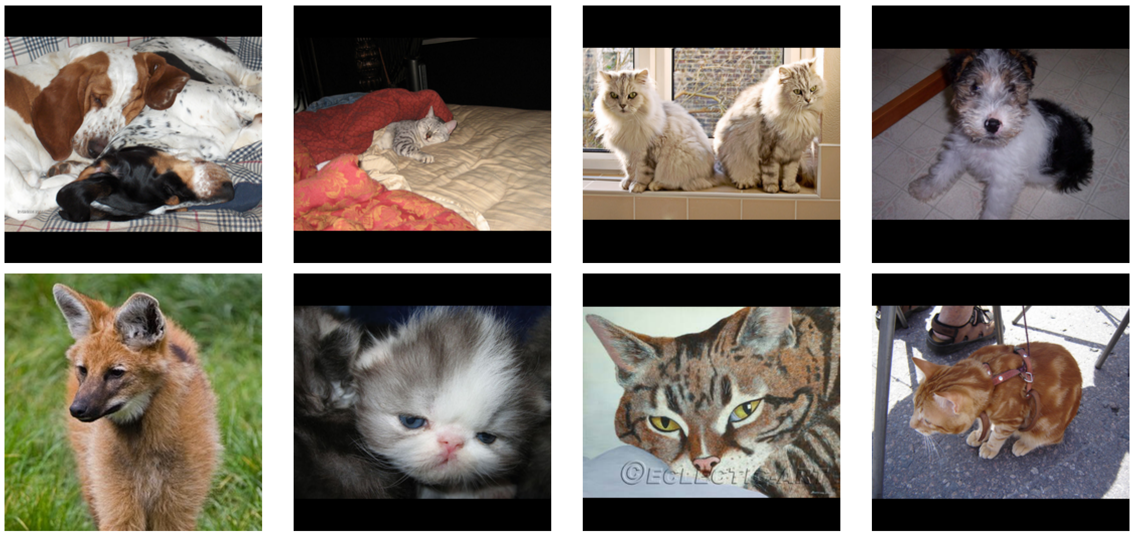

Table 1: Top-8 of most uncertain testing images of our "Dogs and Wolves" dataset according to our HUM CNN

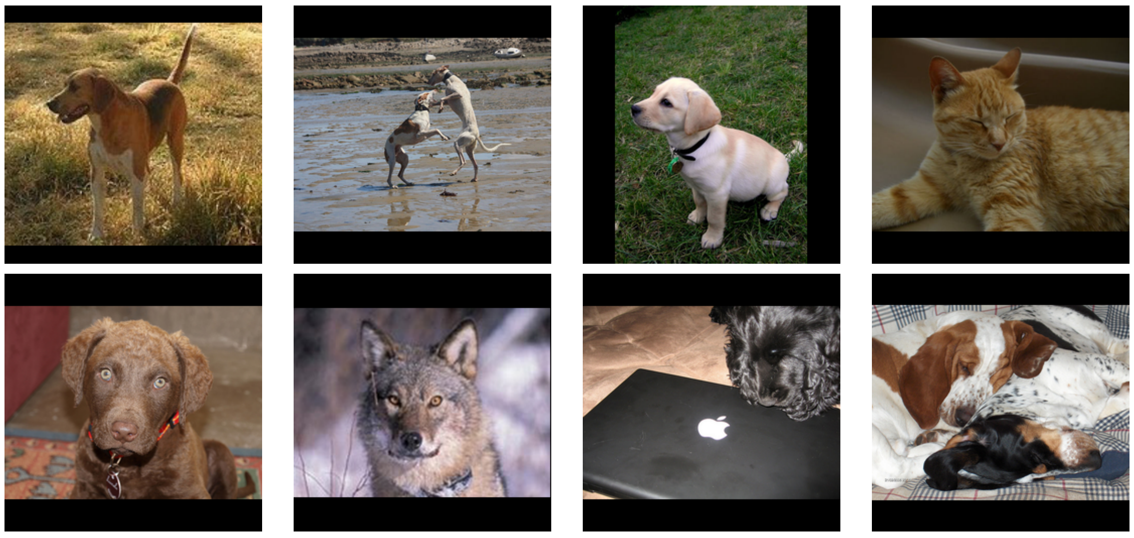

Table 2: Top-8 of most entropic testing images of our "Dogs and Wolves" dataset according to a regular CNN

6 Future Work and Conclusion

In this work, we revisited an old problematic in Machine Learning and more generally in Statistics about dealing with estimation error that is inherent to real-world studied phenomena. An interesting avenue of research could be to refine the "far from dataset" kind of uncertainty: revisiting the Wasserstein distance between the query Dirac and the training dataset would pragmatically give a geometric aspect to uncertainty.

This field of research has a wide of applications: active learning would choose examples based on predicted uncertainty which is intuitively (at least) reasonable: *to discover what we are unsure of first*. There is also another promising application: reinforcement learning without reward to learn how to navigate in a new environement without task which would give some complementary understanding to the notion of curiosity in the scientific effort of Burda et al. [2018] and we could consider that diminishing an uncertainty estimation would be the cornerstone of a reinforcement learning policy. Indeed, even in regular reinforcement learning, training inescapably boils down to distentangling an exploration / exploitation dilemma [Sutton and Barto, 2018]: on the one hand, exploration is testing unusual (or risky) patterns of actions with the hope to expect possibly higher rewards and on the other hand, exploitation is benefiting from known (or conservative) patterns of actions in order to better guarantee a certain level of rewards. This contribution about uncertainty estimation could help breaking the dilemma. The rationale behind could consist in saying that one should explore where uncertainty is high and exploit where uncertainty is low and value estimation is high.

It seems interesting to also investigate the blurry prediction provided by uncertainty models to help for more robustness to adversarial examples (example designed to fool supervisedly trained algorithms, see the survey of Yuan et al. [2019] for details). At least, this kind of systems would be fooled but the predictions would have larger standard deviation for example for this deceiving points.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`.

C. Bishop. Mixture density networks. Technical report, January 1994.

C. M. Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

N. S. Detlefsen, M. Jørgensen, and S. Hauberg. Reliable training and estimation of variance networks. In *Advances in Neural Information Processing Systems*, pages 6323–6333, 2019. URL `http://arxiv.org/abs/1906.03260`.

L. Fei-Fei. Imagenet: crowdsourcing, benchmarking and other cool things. In *CMU VASC Seminar*, volume 16, pages 18–25, 2010.

Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

H. Greenspan, B. Van Ginneken, and R. M. Summers. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging*, 35(5):1153–1159, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. URL `https://arxiv.org/abs/1512.03385`.

S.-J. Huang, R. Jin, and Z.-H. Zhou. Active Learning by Querying Informative and Representative Examples. In *Advances in Neural Information Processing Systems*, 2010.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, number 2014, 2013.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *MIT Press, Cambridge*, 1995.

D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4 (3):448–472, 1992.

D. A. Nix and A. S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

M. Seeger. *Gaussian processes for machine learning.*, volume 14. MIT press Cambridge, MA, 2004. doi: 10.1142/S0129065704001899.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

J. Vermorel. Probabilistic forecasting, 2018. URL `https://youtu.be/KXC-hPCojGQ`.

X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, 2014. ISBN 9783319105895.