Better uncertainty estimates for deep neural networks

Abstract. Nowadays, neural networks are ubiquitous in our everyday lives. Even in sensitive domains such as health sector or finance, we use neural networks to make predictions. In such domains, predictions should be supported by an uncertainty index, to know how confident the machine is with its prediction. Whereas the common method is to use over parametrized networks to predict every thing, recent research has shown it is possible to prune these networks to reduce their storage size and train it faster without affecting the accuracy. Going further in the reflection, since pruning avoid overfitting and should build smoother decision boundaries, it must be possible to get a real improvement in the uncertainty estimates using pruned networks. The main objective of this project is to explore this hypothesis, making some tests on different architectures starting from low complexity datasets and going to real life datasets, We will see that even if we cannot validate the assumption, we take a step forward here.

Hugo Aguettaz
Semester's Project (10 credits ECTS)
Fall 2019
Supervisors: Konstantinos Pitas, Benjamin Ricaud
Professor: Pierre Vandergheynst

Contents

1

Introduction

1.1 The importance of uncertainty estimate

Nowadays, neural networks are used in many domains: healthcare, finance, autonomous driving, art, ... Nevertheless, they proved they are complex to manage and their predictions must be seen with the benefit of hindsight. A neural network can perform well only on data that are distributed in a similar way than the training set distribution. If a neural network is trained to differentiate a dog and a cat, what happens if we give it a platypus image? It must answer so it will but it should also give an idea of its prediction uncertainty. In domain such as healthcare, this information of uncertainty is crucial. Does this patient must consult a specialist based on his retina image? Does he must start some treatment? All depends on the reliability of the prediction given by the neural network. If it is sure of its response, no need to consult a specialist, the patient should start immediately a cure. If not, it's better to consult an expert that can give a confident diagnostic. In this project, I will try to give a better uncertainty estimate for deep neural networks.

1.2 Pruning and uncertainty

To give a better uncertainty estimate, it could be interesting to think about different case and how the uncertainty should varies with this case. Given three different binary classifiers with different fitting regime (1.1) Let's consider the uncertainty as being inversely related to the distance to the classification boundary (the closer from this boundary a point is, the higher the uncertainty is). First of all, it should be intuitive that the appropriate fitting gives the best uncertainty estimate. But what is the difference between under-fitting and over-fitting in this estimate? Well, it all depends on the degree of this regime. A high under fitting as a high over fitting will gives very bad uncertainty estimate. But with a moderate degree, it seems that over fitting is worst because is give very rough boundary and the uncertainty could be very high for some points if this boundary come close to them. Since over-fitting is related to over-parametrised networks as we shall see later, we can think about methods to reduce this number of parameters, that is method to prune a networks. To improve performance on various tasks, number of parameters of neural network architectures had increased exponentially since LeNet5. But these over-parametrised networks reach faster physical limit such as storage or computation. That is why a lot of techniques involved to reduce the number of parameters using pruning methods. Can we also use these methods to obtain a better uncertainty estimates?

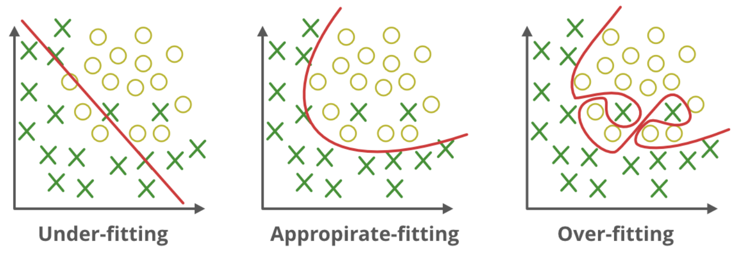


Figure 1.1: Fitting regime

2

Background

2.1 Modern Machine Learning

Machine learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. Algorithms create a model based on a training set to make predictions or decisions without being explicitly programmed to perform the task. Nowadays, deep learning, a subset of machine learning techniques, is omnipresent in the technologies. It requires a large amount of data to perform well but can now get higher performance than human in many tasks such as image classification or trading.

A classifier is trained on a training set such that it minimizes some function $L(x, y)$, typically the empirical risk which corresponds to the error on the training set) to give the best prediction performance in a testing set (typically the true risk). Those classifiers are commonly chosen from some function classes $\mathcal{H}$ and can have different complexity. In the field of neural networks, this complexity is linked to the number of parameters of the network. Practitioners routinely use a huge complexity to fit the training data perfectly or near-perfectly (Alexnet (Krizhevsky et al., 2012), VGG-16 (Simonyan and Zisserman, 2014), Deep Face (Taigman et al., 2014)). From the bias-variance trade-off, it is not possible to minimize both bias and variance. Concretely, by increasing the complexity of a classifier, we increase its variance and reach overfitting. On the contrary, by decreasing the complexity of a classifier, we increase its bias and reach the underfitting. However, practitioners routinely use deep learning methods with a huge amount of data to perfectly fit the training data and finally get very good performance on testing data. A recent paper has reconciled bias-variance tradeoff and the moderne practical by introducing an inductive bias which is better learned with complex architecture (Belkin et al., 2018). This behavior is summarized on the double descent risk curve (figure 2.1).

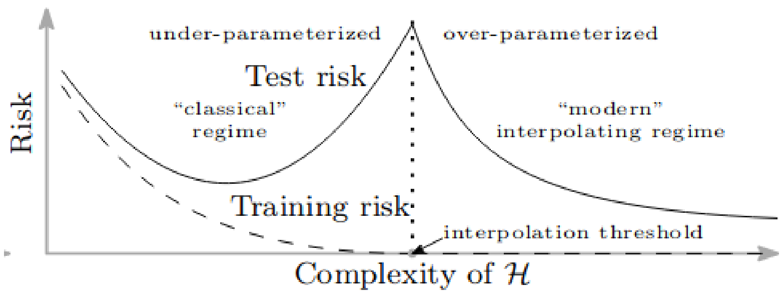


Figure 2.1: Double descent risk curve (Belkin et al., 2018)

2.2 Pruning the network

As discussed previously, the number of parameters has a significant influence on the performance of a neural network. It seems that we always should have as many parameters as possible. But in practice, this is not possible. Over-Parametrized networks are slower to train and harder to store.

But how can we find the optimal architecture, which can get excellent performance with a few parameters? A naive approach should be to have a vague approximation of the required number of parameters. As a role of thumb, with $|W|$ denoting the number of parameters, $|C|$ the number of classes and $|\mathcal{D}|$ the number of training points, we can use the estimate : $|W| = |C| \times |\mathcal{D}|$.

Another approach should be to start with a huge network and to prune it until getting a satisfying number of parameters and accuracy. Such pruning algorithms remove parameters (weights (Han et al., 2015) or filters (Li et al., 2016) according to their magnitude. Both methods use a first pre-training phase where parameters are evaluated to get their real importance in the network. Parameters with the lowest magnitude ($\mathcal{L}_1$-norm) are pruned from the networks. With such method, 90% parameters of VGG-16 have been pruned without affecting the initial accuracy.

The goal of the project is to implement and test different pruning methods for various pruning rates. I only make tests on a weights base pruning method but the filters base pruning method is also implemented. First of all, I wanted to test the influence of the pre-training phase. The Lottery Ticket Hypothesis (Frankle and Carbin, 2019) states that a large neural network will contain a smaller sub-network that, if trained from the start, will achieve a similar accuracy than the larger structure. From this, pre-training should not have a big influence on the performance of the networks. Then, since some connections may compensate other ones but could have an increasing importance with the number of parameters, it could be interesting to test the iterative pruning (that is we prune network by increasing step by step the pruning rate and by reloading the most pruned network for each pruning) versus the non iterative pruning (that is we always reload the full networks and prune on it).

Credits. `https://github.com/wanglouis49/pytorch-weights_pruning`

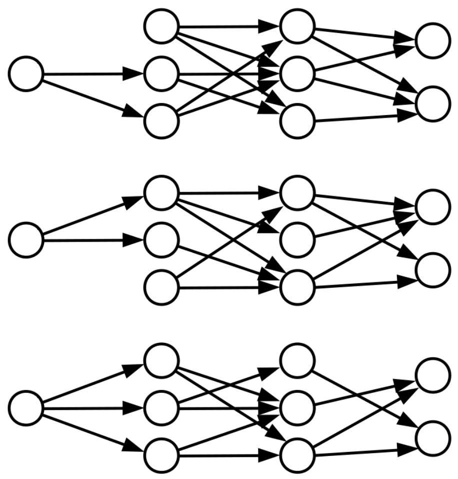
2.3 Uncertainty estimate

On sensitive environments like health, finance or self-driving, it must be interesting to have an idea of the confidence of a given prediction. How much certain is a predictor of its prediction? In the domain of active learning, uncertainty estimate has an important place since the machine asks for information it does not sure of (Geifman and El-Yaniv, 2018).

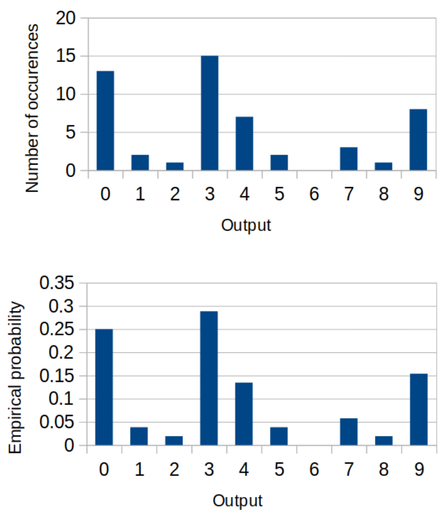
To make such estimates, it should be good to add a probabilistic view of a neural network. A first method could be to imagine random instead of deterministic networks. Weights are now random variables, following some probability density functions (e.g. Gaussian or Laplacian). Using simple probability density functions, characterized by only a few moments (two first moments for Gaussian for example), the number of parameters is only twice the initial one. Optimization can be made using bayesian backpropagation, and optimal parameters can be found (Blundell et al., 2015). It should be possible to make some uncertainties estimate by taking different samples of parameters according to their probability law, resulting in different output from the same input. Nevertheless, this method does not give good results and I did not keep them.

Another approach is to add the probabilistic side using dropout on the testing phase. Using Monte-Carlo simulation by dropout can be seen as a Bayesian approximation (Gal and Ghahramani, 2015). By removing randomly but with a low probability some connections of the networks, the output is random and using Monte Carlo simulation we can derive the empirical probability mass function over the space of possible outputs. Once we have this p.d.m, we can use a various measure of uncertainty. The Shannon entropy is the most intuitive one since it has all the properties we would like for an uncertainty measure (more details in A). The process is described in figure 2.2.

Credits. `https://github.com/OATML/bdl-benchmarks/tree/alpha/baselines/diabetic_retinopathy_diagnosis/mc_dropout`



(a) Networks with randomly dropped connections predict samples of outputs.



(b) From the sample of outputs, an empirical pmf and then an uncertainty estimate are computed

Figure 2.2: Illustration of the Monte Carlo Dropout technique for uncertainties estimate

2.4 Visualisation in adversarial example context

One of the main difficulty when dealing with neural networks is the level of abstraction. To better understand what happened close to the decision boundaries, it could be interesting to show the evolution of the predicted class on images projected on a space generated by two orthogonal vectors which form our basis (Moosavi-Dezfooli et al., 2019). Finding such a basis to project our images on is the last part of my project.

To understand the method, we first have to have a look at adversarial examples. These images are computed to trap a neural network. By adding an imperceptible noise (for the human eye), it will make the predictor predicts wrong. For this, we have to compute the transformation of the image which will increase the highest the loss function $L(h_\theta(x), y)$ where $h_\theta(x)$ is the output given by our predictor of parameters θ for the input x. Thus, we have to compute the gradient of this loss function w.r.t the input. Since we are only interested in the direction to move each pixel to get a wrong classification, we only keep the sign of the gradient. Thus adversarial example becomes :

$$x_{adv} = x + \varepsilon \cdot \text{sign}\left\{\nabla_x L(h_\theta(x), y)\right\}$$

To compute our basis, we use similar reasoning. Here again, the goal is to see the influence of variation on an image will induce the highest change in the loss function. The change is given by the direction of the same gradient than for adversarial example. This direction will be used as the first vector of our basis. The second vector is chosen randomly, from a Gaussian distribution with the same mean and std as the first one. By using the complementary, we enforce the two vectors being orthogonal. Finally, we normalize our vectors to obtain our basis $\mathcal{B}_{u,v}$ such that :

$$\vec{u} = \text{sign}\left\{\nabla_x L(h_\theta(x), y)\right\} \qquad v_i \sim \mathcal{N}(\mu_u, \sigma_u) \quad \forall i$$

All the images on the space generated by our basis are now constructed using :

$$\overrightarrow{I_{ij}} = \overrightarrow{I_0} + c_i \vec{u} + c_j \vec{v} \qquad \text{with } ||\vec{u}|| = ||\vec{u}|| = 1 \text{ and } \vec{u} \perp \vec{v}$$

To compute the gradient, we can use two methods. The Fast Gradient Sign Method (Goodfellow et al., 2015) can be used to get very fast computation. Nevertheless, using a slower method, we can get a better estimate of the gradient. This is the case of the Deep Fool method (S.M Moosavi-Dezfooli, 2016) which compute components by components the gradient.

Credits. `https://github.com/1Konny/FGSM` and `https://github.com/LTS4/DeepFool`

3

Experiments

Remark. Implementation of simulations was made on Jupyter Notebook using a Python3 interpreter and the neural network's module PyTorch. Simulations ran on a GeForce Ti1080 GPU with 11GB of RAM through the server lts2gdk0.epfl.ch.

3.1 Assumptions

The goal of theses experiments was to evaluate the quality of the uncertainty estimate and compare these evaluations between a complete network and a pruned network. To rank the quality of an uncertainty estimate, we are interested in two criteria, based on some assumptions :

1. Assuming the uncertainty will be higher for the data points close to a classification boundary, the data points with high accuracy are more likely to be misclassified. Thus, if we keep only 50% of the data points with the lowest uncertainty estimate to make a prediction, the accuracy will be higher than if we keep all data points.
2. Assuming that two data points are correctly classified, if we interpolate linearly between these points, the uncertainty should be maximum between them and minimum on these points.
3. Assuming the margin (the distance from a point to the closest decision boundary) being related to the uncertainty estimate, our network which has the best uncertainty estimate should have the highest margin.

3.2 Preliminary experiments

To begin with, it is interesting to make some experiments with synthetic datasets where simulations are very fast to run. It consists of a 3 classes well balanced dataset, in a two dimensional space (figure 3.1). The train, validation and test sets contains respectively 3'000, 1'000 and 1'000 points.

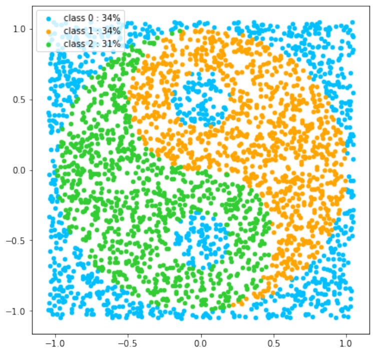


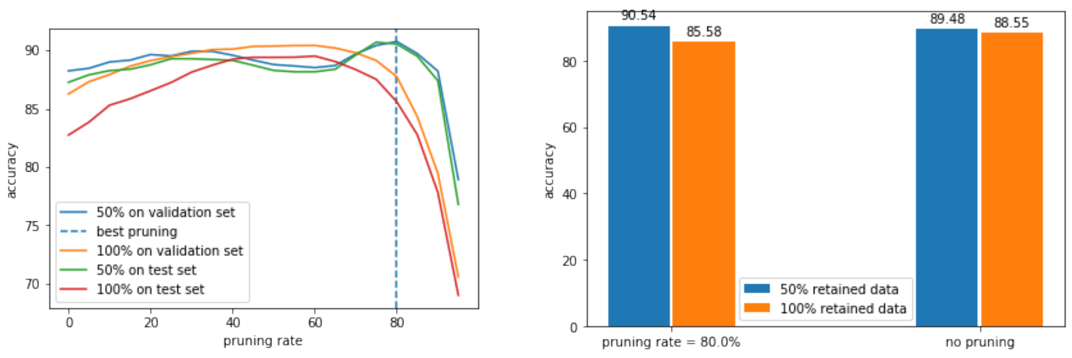
Figure 3.1: Synthetic dataset with 3 classes

The goal of this experiment is to test our algorithm and to have a first look on our hypothesis. On the first part, we test different pruning techniques and see if there exists a potential improvement in the uncertainty estimate for both techniques. I evaluated three different pruning techniques' performance :

- **Net1.** The network is pre-trained on 50 epochs to find which connections are important, and which can be pruned. Next, it is pruned by iteration, with a pruning step of 1%, from 1% to 99% of pruned parameters. After each pruning, the network is re-trained on 10 epochs.
- **Net2.** For the first step, the network is pruned randomly, without pre-training. Afterwards, the network is pruned by iteration, in the same vein than the previous network.
- **Net3.** The network is pre-trained on 50 epochs to find the relevant connections. Then, we apply one-shot pruning, for all pruning rate between 1% and 99%. After each pruning, the network is re-trained on 10 epochs.

Experiment 1. Uncertainty estimate and pruning rate.

I started to compare the accuracy on the 50% of data with the lowest uncertainty to all data, for each pruning techniques. A network trained on 100 epochs, is stored and used as reference. The optimal pruning method is choose as the one which give the best accuracy on 50% of retained data on the validation set. The performance of the best pruned network is compared to the reference. Results are presented on figure 3.2.



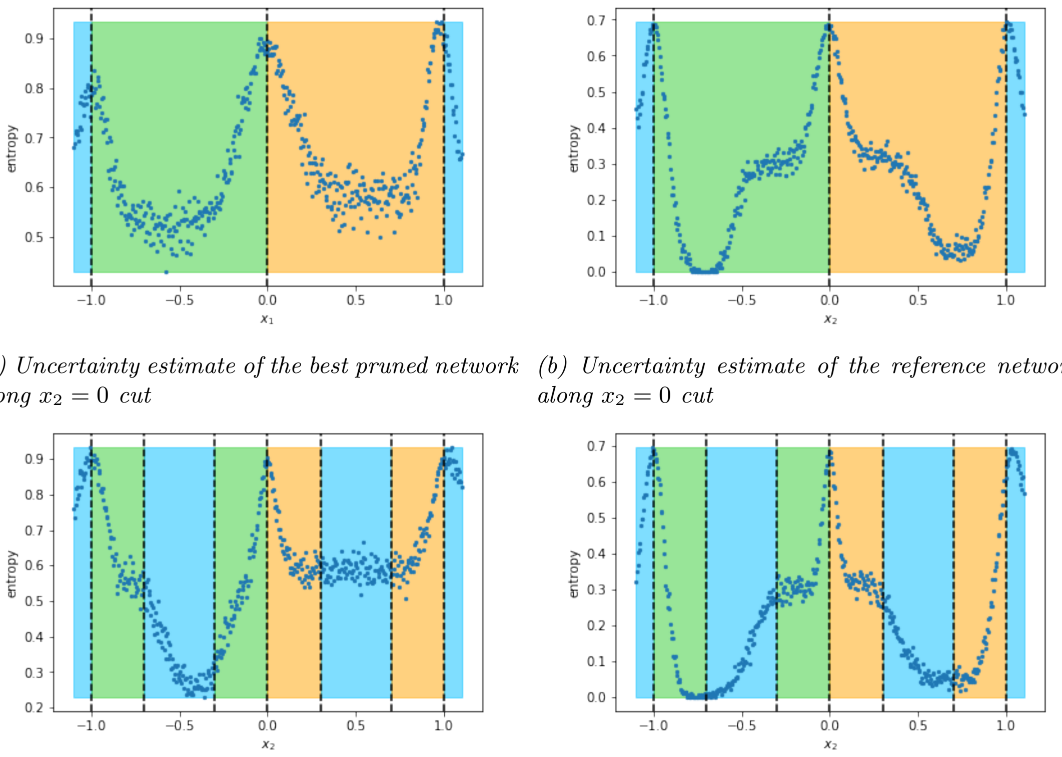
(a) Evolution of the accuracy depending on the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets

(b) Bar plot of the accuracy on the testing set after keeping all data or the 50% with lowest uncertainty on the reference network and the best pruned network

Figure 3.2: Accuracy depending on the uncertainty estimate for best pruned network and reference network

Experiment 2. Uncertainty estimate along sectional views.

To observe the variation of the uncertainty over data points, I also prospected the uncertainty along two sectional views. The goal is to see the evolution of the uncertainty near borders between different classes. I drew uncertainty evolutions along x axis (respectively y axis) at $y = 0$ (respectively $x = 0$) and displayed them in figure 3.3.



(a) Uncertainty estimate of the best pruned network along $x_2 = 0$ cut

(b) Uncertainty estimate of the reference network along $x_2 = 0$ cut

(c) Uncertainty estimate of the best pruned network along $x_1 = 0$ cut

(d) Uncertainty estimate of the reference network along $x_1 = 0$ cut

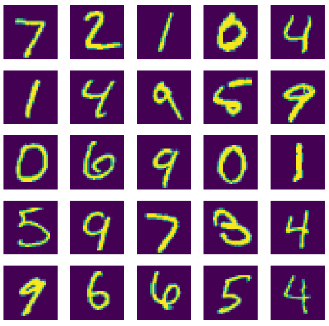
Figure 3.3: Uncertainty along horizontal and vertical cuts for the best pruned network and the reference network

3.3 Classical experiments

Classical experiments are made on MNIST, Fashion MNIST, and CIFAR-10 datasets. Samples of these datasets are shown in figure 3.4. We construct one fully connected and one convolutional network for each dataset and design them in order to have around 100K parameters for 2 hidden layers. I used ReLU as activation function between each layer and a softmax output. For each dataset, a network is trained on 100 epochs and the one which obtains the best performance on the testing set overall training epochs is saved. I used it as a reference network to evaluate my method. Then, other networks are built, according to the pruning methods presented in section 2.2 :

- **Net1.** The network is pre-trained on 30 epochs in order to find which connections are important, and which can be pruned. Next, it is pruned by iteration, with a pruning step of 5%, from 5% to 95% of pruned parameters. After each pruning, the network is re-trained on 10 epochs.
- **Net2.** For the first step, the network is pruned randomly, without pre-training. Afterward, the network is pruned by iteration, in the same vein than the previous network.
- **Net3.** The network is pre-trained on 30 epochs in order to find the relevant connections. Then, we apply one-shot pruning, for all pruning rate between 5% and 95%. After each pruning, the network is re-trained on 10 epochs.

In order to minimize the effect of the random initialization on our results, we save 10 networks with 10 different initializations for both pruning methods and reference.


(a) MNIST sample

(b) Fashion-MNIST sample

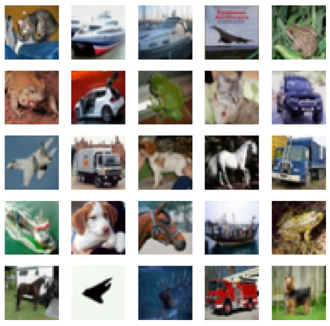

(c) CIFAR-10 sample

Figure 3.4: 25 images on each classical dataset

Experiment 1. Uncertainty estimate and pruning rate.

First of all, we would like to see if there is an improvement in term of the uncertainty estimates by pruning a network. For this, for each technique of pruning, we start by selecting the pruning rate which gives the highest accuracy on a validation set. Then, we keep 50% of the data from the testing set with the lowest uncertainty and compute the accuracy of the pruned networks and the reference one on these points. The accuracy overall testing set is also shown. Synthetic results are presented in table 3.1. For more details, please refer to section B.

Dataset	Type	Name	Accuracy on 50% data	Accuracy on 100% data
MNIST	DenseNet	Reference	99.23 ± 0.13	96.02 ± 0.16
		Net1	99.66 ± 0.11	96.9 ± 0.08
		Net2	99.58 ± 0.10	96.54 ± 0.12
		Net3	98.62 ± 0.24	93.14 ± 0.39
	ConvNet	Reference	99.18 ± 0.11	99.07 ± 0.07
		Net1	99.44 ± 0.07	99.01 ± 0.05
		Net2	99.44 ± 0.11	99.01 ± 0.06
		Net3	99.04 ± 0.12	98.94 ± 0.08
Fashion-MNIST	DenseNet	Reference	93.63 ± 0.53	85.98 ± 0.11
		Net1	95.97 ± 0.56	86.98 ± 0.15
		Net2	95.26 ± 1.39	86.07 ± 1.81
		Net3	92.71 ± 1.82	83.56 ± 0.99
	ConvNet	Reference	92.43 ± 0.30	90.32 ± 0.13
		Net1	94.49 ± 0.25	90.72 ± 0.16
		Net2	93.98 ± 0.25	90.35 ± 0.15
		Net3	89.9 ± 0.35	87.78 ± 0.29
CIFAR-10	DenseNet	Reference	58.93 ± 0.56	47.85 ± 0.24
		Net1	59.91 ± 0.32	48.00 ± 0.33
		Net2	60.32 ± 0.67	48.21 ± 0.32
		Net3	56.93 ± 0.57	46.34 ± 0.21
	ConvNet	Reference	71.18 ± 0.50	66.00 ± 0.23
		Net1	73.41 ± 0.41	66.74 ± 0.24
		Net2	71.87 ± 0.66	65.80 ± 0.43
		Net3	65.35 ± 0.76	61.09 ± 0.46

Table 3.1: Accuracy (mean ± std) for the different neural networks of experiment 1.

Experiment 2. Uncertainty estimates along with interpolation.

In the second part of the experiment, I would like to test the uncertainty estimate along interpolation between pairs of images. Let I_1 and I_2 be two images from the testing set. I can evaluate the uncertainty estimate for any image I_α between I_1 and I_2 and defined as :

$$I_\alpha = \alpha \cdot I_1 + (1-\alpha) \cdot I_2 \qquad \forall \alpha \in [0,1]$$

In the same way than the previous experiment, I kept the best-pruned network of each method. Since I evaluated the uncertainty on 100 pairs of images (10 pairs for each of the 10 initializations), it was relevant to use non-parametric regression with a Gaussian kernel in order to make the interpretation easier. Since the experiment is highly correlated to the first one and will not give meaningful contribution to the project, I only present results for the best network overall techniques for each dataset/network. The results are presented in section B.

Experiment 3. Decision boundaries and pruning rate.

My last experiment was to focus on decision boundaries built by the networks. According to the method presented in section 2.4, I displayed these boundaries around 25 images from each dataset/network (see B. The green area represents the true class while the red areas are wrong classes. Black lines are frontiers between 2 classes. Since the results were not easy to interpret using this visualization, I also compute the margin of each 25 images (the distance to the closest border) and for all initializations, for all dataset/network. The mean and standard deviations of theses margins are presented on table 3.2.

Dataset	Type	Name	Margin
MNIST	DenseNet	Reference	1.32 ± 0.87
		Net1	1.42 ± 0.80
		Net2	1.49 ± 0.90
		Net3	1.70 ± 1.17
	ConvNet	Reference	2.65 ± 1.10
		Net1	3.00 ± 1.25
		Net2	3.12 ± 1.26
		Net3	2.71 ± 1.12
Fashion-MNIST	DenseNet	Reference	0.92 ± 0.79
		Net1	0.80 ± 0.74
		Net2	0.95 ± 0.88
		Net3	1.39 ± 1.08
	ConvNet	Reference	1.02 ± 0.73
		Net1	0.90 ± 0.69
		Net2	0.97 ± 0.78
		Net3	1.25 ± 1.06
CIFAR-10	DenseNet	Reference	0.42 ± 0.52
		Net1	0.34 ± 0.49
		Net2	0.38 ± 0.57
		Net3	0.53 ± 0.80
	ConvNet	Reference	0.55 ± 0.59
		Net1	0.49 ± 0.51
		Net2	0.50 ± 0.49
		Net3	0.57 ± 0.63

Table 3.2: Margin (mean $\pm$ std) for the different neural networks of experiment 3.

3.4 Advanced experiments

To finish, I tried to implement my method on a more advanced dataset: the Diabetic Retinopathy Diagnosis dataset. It is composed of retina images 3.5 and classified as healthy or diseased, according to `https://github.com/OATML/bdl-benchmarks` and (Leibig et al., 2017).

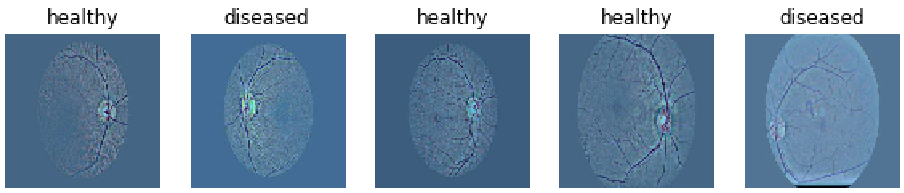


Figure 3.5: Retina dataset with 2 classes : healthy (class 0) or diseased (class 1)

Such advanced dataset is harder to deal with. To begin, since the images are huge ($3 \times 256 \times 256$), and the network contains a lot of parameters (more than 1M, see figure 3.6), I properly monitored the memory allocation. I ensure to not store useless data, such as graphs or networks during the testing phase. I reduced the batch size to 50 images to permit the network being trained. I also had to use the server of the LTS2 which is equipped with a Graphic Card GeForce GTX 1080 Ti 11Go.

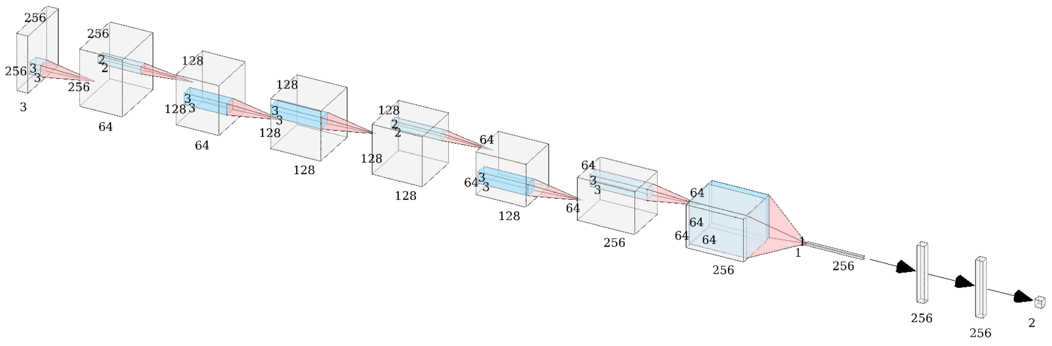


Figure 3.6: Retina network architecture

Second, due to the complexity of the dataset, increase the number of parameters was not sufficient, I also increase the number of epochs for both the training of the reference network and the pretraining and retraining of the pruned network. To make trained networks more robust, I decided to use k-fold like's method for training and testing. To finish, unlike the previous experiments, for each network, I stored the one with the best performance on the testing set.

Last but not least, the main issue of this dataset was that it was not well balanced. It contains around 80% healthy retina against 20% diseased retina. To counteract this, I had to make some change on my method. First of all, I did not use the accuracy anymore but the f1 score which gives the best indication on such case. Since this measure is very versatile and very sensitive to misclassified, it was not useful but required to train on many epochs and to record only the best networks. I also use a weighted class loss function (Cui et al., 2019) instead of the classic one. This method avoids the network to learn only to predict the most represented class using different weight depending on the true class of all points. Since the healthy class is 4 times more represented than the diseased class, I weighted the loss to make 4 healthy images have the same impact as 1 diseased image.

In the same way, as for the preliminary experiment, I tried to test the influence of pruning connections on the uncertainty estimate and accuracy of the network. Because network and dataset are huge and computations take more time, I have not been able to test different pruning methods and different initializations to remove the random effect. I only evaluated the best-pruned network, a Net1's network, based on its performance on 50% of the data with the lowest uncertainty and compare it to the reference network which gets the highest f1 score overall training epochs. The results are shown in figure 3.7

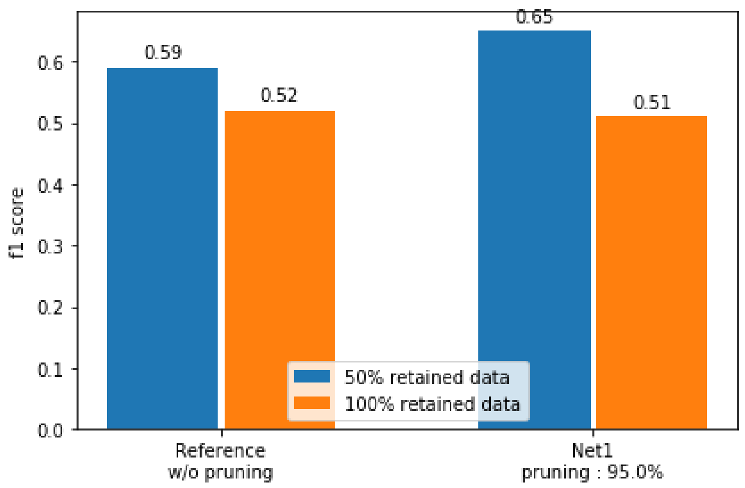


Figure 3.7: F1 measure while keeping all data or the 50% with the lowest uncertainty on testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

4

Discussion

4.1 Preliminary experiment

Experiment 1. Uncertainty estimate and pruning rate.

First of all but without illustrations, Net1 is the one which gives the best result among all pruning methods. For this very reason, I only show results for this pruning method. The best pruning rate is chosen based on accuracy on 50% of data with the lowest uncertainty on the validation set. As we can see in figure 3.2, trend on validation and test sets are similar. It is the first fact that we will see on other data sets and that we can now assume as correct. My 10K parameters' network can be pruned up to 80% without loss of accuracy. What is more, it would seem that pruning the network permits to have a better uncertainty estimate since the accuracy on the 50% data with the lowest uncertainty seems to have a peak around 80% of pruning whereas the gap with the accuracy on all data is higher. If the most uncertain points were directly the most misclassified points, that exactly what would happen. Observing the histogram, a similar guess could be stated. Indeed, the pruned network seems to compensate its poorer performance on all data by handing a better uncertainty estimate than the reference one and that enables it to get the best performance on the 50% with the highest certainty.

Experiment 2. Uncertainty estimate along sectional views.

Now, observing the results on 3.3, the uncertainty estimate difference between pruned and reference network seems to not be as obvious as on the previous experiment. Patterns look similar for both networks: the uncertainty is higher on borders, lower in the middle of a class, with some exception (frontier between blue and green classes along with the $x_1 = 0$ cut). But we can also observe that if the uncertainty goes to 0 for some points with the reference network, it never reach this "perfect" prediction for the pruned network. It is as if the pruned network would not tell "I'm sure of my prediction". In fact, this can be explained as follow: since uncertainty estimate is made by removing random connections, the network with a few but very useful connections should be more affected than the one with a huge amount of more or less important connections. As a result, the pruned network can't be sure of its prediction since different Monte Carlo simulations lead to different outputs to the same input.

4.2 Classical experiments

Experiment 1. Uncertainty estimate and pruning rate.

From table 3.1, we can observe and say many things. In the first place, even if the accuracy of the network decreases when the complexity of the dataset increases, the gap between the accuracy for 50% of retained data and all retained data stays constant. The accuracy on MNIST is too high to let results being significant with or without using uncertainty estimates. Secondly, we can observe that pruning connections do not reduce the accuracy of the network since, for all network and all experiments, the pruned network accuracy is more or less 2% around the reference one.

However, it seems that our two first methods (Net1 and Net2) are significantly more efficient than the third one. The third method gives always poorer results than the reference one whereas the two first are as good or better than the reference, for both 50% or 100% retained data. Nevertheless, in total, Net3 is trained on many fewer epochs than the other ones (40 epochs against more than 100 epochs). For this reason, the comparison is not so meaningful. It could be interesting to increase the number of re-train after the one-shot pruning to more than 50 epochs or to use early stop criterion during re-training. The difference between Net1 and Net2 is not obvious. Indeed, even if Net1 gives better results than Net2 on MNIST and Fashion-MNIST, the trend reverses on CIFAR-10. It could be interesting to compare these two methods on more complex datasets to see. Concerning the robustness of our methods, it is not so easy to conclude something. Standard deviations for each methods are very close. Actually, pruning connections seems to not give more robustness in comparison to the reference network, but it seems that some pruning method is more robust than other (Net1 against Net2 and Net3 even if the difference is not obvious for all datasets and network's types.

Experiment 2. Uncertainty estimates along interpolation.

In the same vein that I claimed it in the preliminary experiment, the uncertainty estimate along interpolation is highest for pruned networks than for the reference network. It the case for almost all pruning techniques, data sets and network's types. As explained before, it can be a consequence of the higher magnitude of connections (and then importance) of pruned networks. It should be interesting to observe those magnitudes for the different network to validate this hypothesis. Next, the uncertainty along the interpolation for the pruned network is more in line with my second assumption in section 3.1. There exists at least two possible scenarios. Either this assumption is wrong even if it appears to be of common sense or pruning gives better uncertainty estimates as we stated it in our research hypothesis. Nevertheless, the difference is not significant for all data-sets and networks. First, when the complexity of the network is too high and the network does not get enough performance, the uncertainty can not be well estimated because the network is not even confident with its predictions on initial images. Next, the interpolation appears to give better results for DenseNets than for ConvNets. Maybe the method of pruning ConvNet should be modified for example using the filters based pruning method presented in section 2.2.

Experiment 3. Decision boundaries and pruning rate.

Neither visualization of decision boundaries nor the table of margins (table 3.2 are easy to interpret. In fact, overall networks, there is no one with the best margin. The standard deviation is too high to allow a correct reading of results. Even if this visualization method gave good result in adversarial example context (Moosavi-Dezfooli et al., 2019), perhaps it is not well adapted to my project and I should use other dimensionality reduction methods such as PCA or using other features' extraction techniques. Moreover, instead of using margin I also could use some curvature measures around data points and see if the pruned network gives flatter decision boundaries. This could be done in further experiments.

4.3 Advanced experiments

For this data set, I can not say many things from results. Even if I am very tempted by saying our pruning method leads to higher performance, I can not conclude. Indeed, I only tested one pruning method and for one initialization. Thus, results are under the influence of randomness and the training was not optimal. It should be better to train on more epochs and to test on many seeds to get more robustness in our performance measurements. I just finish the discussion saying that the Diabetic Retinopathy Diagnosis benchmark obtained for its 4M parameters network an accuracy around 92% for 50% retained data and 85% for all data (hard to compare with our F1 measure).

5

Conclusion

5.1 Reached objectives

Even if the hypothesis can not be validated for sure, my results seem to be in line with it. First of all, I showed that it is possible to get very good results keeping only 20% most important connections, just as it has been shown in many papers. Nevertheless, the choice of the pruning method is critical and all training parameters. In future research, it could be interesting to optimize current techniques (increase the number of re-training epochs for the one-shot pruning) but to test different pruning method such as the filters' based one. This could give better results for ConvNets in particular.

Concerning our research question, it appears to have slight improvements using pruning in the uncertainty estimate. Further research could be to evaluate our method and derivatives with other common architectures (VGG-16, ResNet, GoogLeNet,...) on more complex data-set (ImageNet, ...). It should also be interesting to derive other tests such as interpolation one to evaluate the quality of the uncertainty estimate. Different uncertainty evaluation method should also be tested, using, for example, bayesian networks (see section 2.3 to get new regard on the uncertainty. Our Monte Carlo Dropout method should also be optimized for example by computing dropout probability depending on the number of parameters in each layer.

Last, to better understand what happens in neural networks, it should be interesting to improve visualizations' techniques. This could be a determinant factor in the approval or rejection of our hypothesis and a significative step in the domain.

5.2 Project

First of all, thank you for this project and for supervising me. I have the feeling to learn a lot of things in many domains. Naturally, I improved my understanding of Deep Learning and got new regard, more probabilistic on neural networks. Trying to deal with a simple human concept such as uncertainty has been very interesting for me and I started to see the limitations of such technologies. I also learned a lot of things in machine learning, information theory and statistics method.

Being independent in my work was very instructive, I applied my knowledge in probability theory and statistics to this project and it was very satisfying. Equilibrium between theory and practice, I find this project well balanced. The theoretical background to understand the concepts was sufficiently complex to pique curiosity but not enough to be lost. Since PyTorch is now a reference in the Deep Learning domain, finding concept's implementation was easy but changing it to our purpose required to have a good understanding of the code.

Bibliography

M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine learning and the bias-variancetrade-off, 2018. URL https://arxiv.org/pdf/1812.11118.pdf.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks, 2015. URL https://arxiv.org/pdf/1505.05424.pdf.

Y. Cui, M. Jia, T. Lin, Y. Song, and S. Belongie1. Class-balanced loss based on effective number of samples, 2019. URL https://arxiv.org/pdf/1901.05555.pdf.

J. Frankle and M. Carbin. The lottely ticket hypothesis: Finding sparse trainable neural networks, 2019. URL https://arxiv.org/pdf/1803.03635.pdf.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015. URL https://arxiv.org/pdf/1506.02142.pdf.

Y. Geifman and R. El-Yaniv. Deep active learning with a neural architecture search, 2018. URL https://arxiv.org/pdf/1811.07579.pdf.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015. URL https://arxiv.org/pdf/1412.6572.pdf.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficientneural networks, 2015. URL https://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks, 2012. URL https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

C. Leibig, V. Allken, M. S. Ayhan, P. Berens, and S. Wahl. Leveraging uncertainty information from deep neural networks for disease detection, 2017. URL https://www.nature.com/articles/s41598-017-17876-z.pdf.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets, 2016. URL https://arxiv.org/pdf/1608.08710.pdf.

S. Moosavi-Dezfooli, J. Uesato, A. Fawzi, and P. Frossard. Robustness via curvature regularization, and vice versa, 2019. URL http://openaccess.thecvf.com/content_CVPR_2019/papers/Moosavi-Dezfooli_Robustness_via_Curvature_Regularization_and_Vice_Versa_CVPR_2019_paper.pdf.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. URL https://arxiv.org/pdf/1409.1556.pdf.

P. F. S.M Moosavi-Dezfooli, A. Fawzi. Deepfool: a simple and accurate method to fool deep neural networks, 2016. URL https://arxiv.org/pdf/1511.04599.pdf.

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification, 2014. URL https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Taigman_DeepFace_Closing_the_2014_CVPR_paper.pdf.

Appendix A

Entropy

In this project, we spoke a lot about entropy (Shannon entropy) as a measure of uncertainty. In this section, I stand the main definitions of Shannon entropy and derive some good properties for an uncertainty measure.

A.1 Definition

Shannon entropy. Let X be a discrete random variable taking value in C and admitting pdm $\mathbb{P}(\{X = x\}) = p_x$. We define the Shannon entropy as :

$$S(X) = -\sum_{x \in C} p_x \log p_x \tag{A.1}$$

Joint Shannon entropy. Let $X_1, \ldots, X_n$ be a collection of random variables taking value in the same space C and admitting pdm.s $\mathbb{P}(\{X_i = x_i\}) = p_{x_i}, i = 1, \ldots, n$. We define the joint Shannon entropy as :

$$S(X_1, \ldots, X_n) = -\sum_{(x_1, \ldots, x_n) \in C^n} p_{x_1 \ldots x_n} \log p_{x_1 \ldots x_n} \tag{A.2}$$

Conditional Shannon entropy. Let $X_1, \ldots, X_n$ be a collection of random variables taking value in the same space C and admitting pdm.s $\mathbb{P}(\{X_i = x_i\}) = p_{x_i}, i = 1, \ldots, n$. Let l be a non negative number such that $l = n - m$. We define the conditional Shannon entropy as :

$$\begin{aligned} & S(X_1, \ldots, X_m | X_{m+1}, \ldots, X_n) \\ = -\sum_{(x_{m+1}, \ldots, x_n) \in C^l} \sum_{(x_1, \ldots, x_m) \in C^m} & p_{x_1 \ldots x_m} \cdot \log p_{x_1 \ldots x_m | x_{m+1} \ldots x_n} \end{aligned} \tag{A.3}$$

Moreover, if the collection $X_1 \ldots X_m$ is independent of the collecton $X_{m+1} \ldots X_n$, we have :

$$S(X_1, \ldots, X_m | X_{m+1}, \ldots, X_n) = S(X_1, \ldots, X_m) \tag{A.4}$$

A.2 Properties

Chain rule for sequence of random variables. Let $X_i, i \in \{1 \ldots n\}$ be a collection of random variables. Then we have the following property :

$$S(X_1, \ldots, X_n) = \sum_{i=1}^{n} S(X_i | X_{i-1}, \ldots, X_1) \tag{A.5}$$

Proof. Let $X_i, i \in \{1 \ldots n\}$ be a collection of discrete random variables taking value in C. We define the marginal pmf.s as $\mathbb{P}(\{X_i = x_i\}) = p_{x_i}$. We also pose $0 \leq m \leq n$ and $l = n - m \geq 0$. Thus :

$$\begin{aligned}
S(X_1,\dots,X_n) &= -\sum_{x_1\dots x_n\in C^n} p_{x_1\dots x_n}\log p_{x_1\dots x_n}\\
&= -\sum_{x_1\dots x_{n-1}\in C^{n-1}}\sum_{x_n\in C} p_{x_1\dots x_n}\log p_{x_1\dots x_n}\\
&= -\sum_{x_1\dots x_{n-1}\in C^{n-1}}\sum_{x_n\in C} p_{x_1\dots x_n}\log p_{x_1\dots x_{n-1}}p_{x_n|x_1\dots x_{n-1}}\\
&= -\sum_{x_1\dots x_{n-1}\in C^{n-1}}\sum_{x_n\in C} p_{x_1\dots x_n}\log p_{x_1\dots x_{n-1}} + p_{x_1\dots x_n}\log p_{x_n|x_1\dots x_{n-1}}
\end{aligned}$$

But since :

$$\begin{aligned}
&-\sum_{x_1\dots x_{n-1}\in C^{n-1}}\sum_{x_n\in C} p_{x_1\dots x_n}\log p_{x_1\dots x_{n-1}}\\
&= -\sum_{x_1\dots x_{n-1}\in C^{n-1}}\sum_{x_n\in C} p_{x_n}p_{x_1\dots x_{n-1}|x_n}\log p_{x_1\dots x_{n-1}}\\
&= -\sum_{x_1\dots x_{n-1}\in C^{n-1}} p_{x_1\dots x_{n-1}}\log p_{x_1\dots x_{n-1}}\\
&= S(X_1\dots X_{n-1})
\end{aligned}$$

And :

$$\begin{aligned}
&-\sum_{x_1\dots x_{n-1}\in C^{n-1}}\sum_{x_n\in C} p_{x_1\dots x_n}\log p_{x_n|x_1\dots x_{n+1}}\\
&= S(X_n|X_1\dots X_{n-1})
\end{aligned}$$

We finally get :

$$S(X_1,\dots,X_n) = S(X_1\dots X_{n-1}) + S(X_n|X_1\dots X_{n-1})$$

By iteration over the above equation, we complete the proof.

Additivity. Let $X_i, i\in\{1\dots n\}$ be a collection of independent random variables. Then we have the following property :

$$S(X_1,\dots,X_n) = \sum_{i=1}^{n} S(X_i) \tag{A.6}$$

Proof. Using the chain rule property with independent random variables, we directly complete the proof.

Maximum entropy. The discrete uniform distribution has maximum entropy over all discrete distributions and $S=\log|C|$.

Proof. First of all, let's recall the inequality of weighted arithmetic mean and weighted geometric mean. Let $x_1,\dots x_n$ be non negative numbers and $w_1,\dots w_n$ be non negative weights such that $\sum w_i = w$. Such we have the following inequality :

$$\frac{1}{w}\sum w_i x_i \geq \sqrt[w]{\prod x_i^{w_i}}$$

With equality if and only if all the x_i are equal with $w_i>0$.

Using the definition of Shannon entropy and the above inequality, we have :

$$\exp(S) = \exp\left(-\sum_{x\in C} p_x \log p_x\right) = \prod_{x\in C}\left(\frac{1}{p_x}\right)^{p_x} \leq \sum_{x\in C} p_x \frac{1}{p_x} = |C|$$

With equality if and only if all $\frac{1}{p_x}$ are equal with $p_x > 0$ which corresponds to the discrete uniform distribution. Thus we complete the proof.

Minimum entropy. The distribution corresponding to an almost sure event has minimum entropy over all distributions with $S = 0$.

Proof. First of all, it is easy to prove $S \geq 0$ for all probability distribution. Using the definition of Shannon entropy, and using the fact that $0 \leq p_x \leq 1$ for all $x \in C$ we get :

$$S = -\sum_{x\in C} p_x \log p_x \geq 0$$

Now let's compute the Shannon entropy of an almost sure event. We have $p_k = 1$ for some k and $p_j = 0$ for all $j \neq k$. Thus, we have :

$$S = -\sum_{x\in C} p_x \log p_x = p_k \log p_k = \log 1 = 0$$

This distribution reaches the minimum of the Shannon entropy and thus we complete the proof.

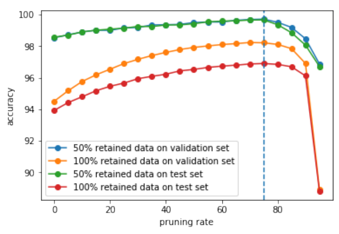
Appendix B

Results

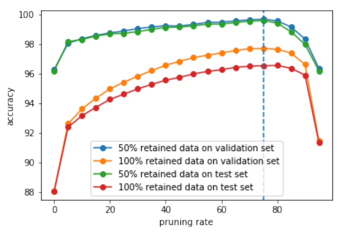
B.1 MNIST

Fully connected neural networks. We use a fully connected neural networks with almost 100'000 parameters. It is composed of two hidden layers with 128 and 64 neurons, a ReLU activation function between each layer and a Softmax activation function for the output layer.

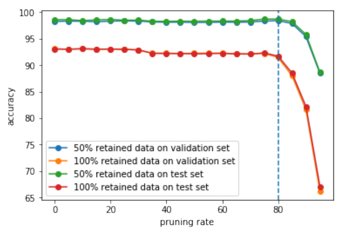
Experiment 1. Accuracy on 50% and 100% data with the lowest uncertainty



(a) Network with pretraining and iterative pruning



(b) Network without pretraining and with iterative pruning



(c) Network with pretraining and with one shot pruning

Figure B.1: Evolution of the accuracy with the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets for the three pruning techniques.

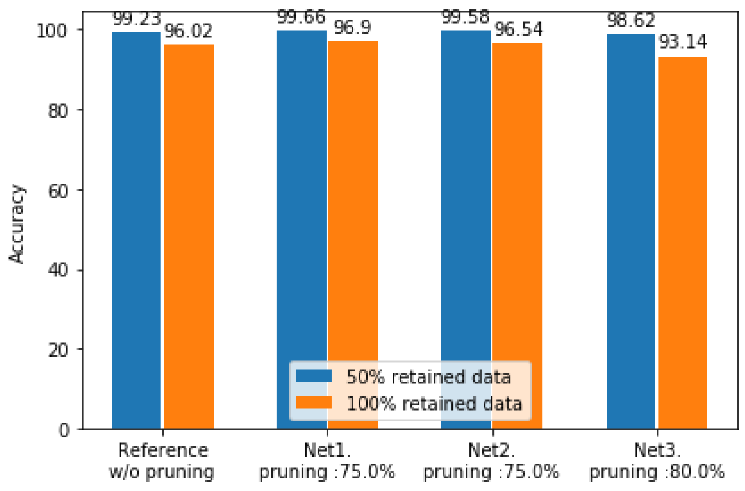
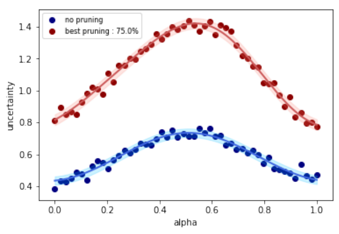
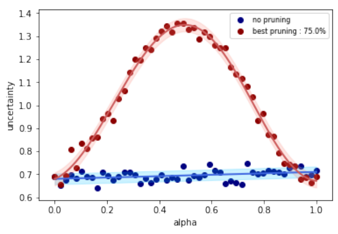


Figure B.2: Accuracy while keeping all data or the 50% with the lowest uncertainty on testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

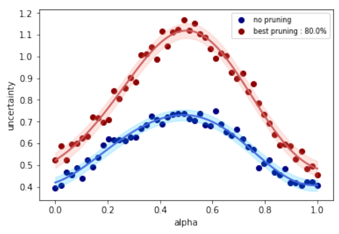
Experiment 2. Uncertainty estimates along the interpolation between pairs of well classified images.



(a) Network with pretraining and iterative pruning



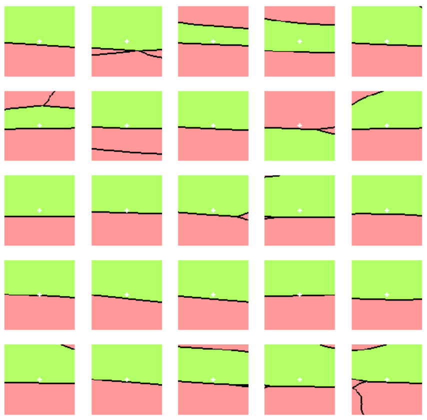
(b) Network without pretraining and with iterative pruning



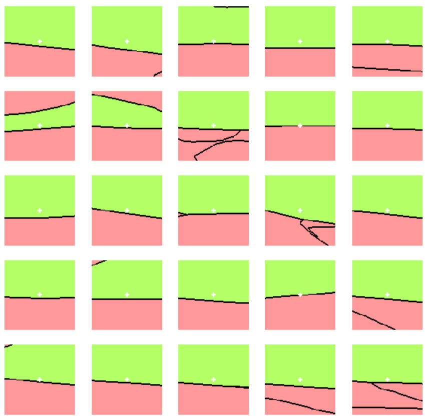
(c) Network with pretraining and with one shot pruning

Figure B.3: Evolution of the uncertainty estimate along the interpolation between 100 pairs of images taken in the test set.

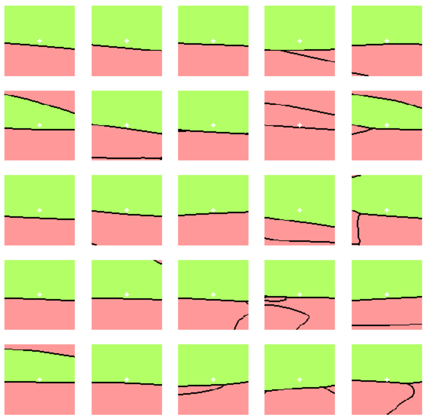
Experiment 3. Visualisation of decision boundaries around 25 images.



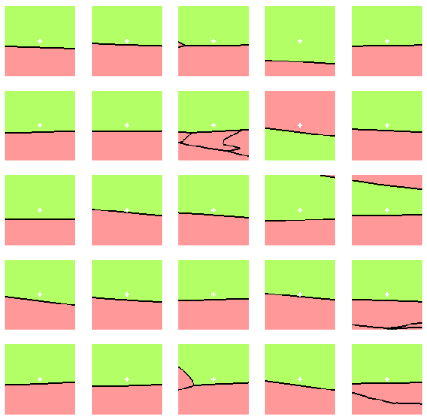
(a) Reference network



(b) Network with pretraining and iterative pruning



(c) Network without pretraining and with iterative pruning

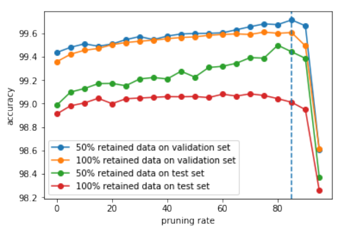


(d) Network with pretraining and with one shot pruning

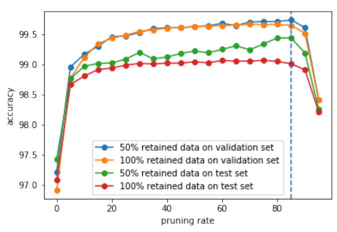
Figure B.4: Decision boundaries for different networks.

Convolutional neural networks. We use a convolutional neural networks with almost 100'000 parameters. It is composed of two hidden layers with 64 channels kernel 5 by 5 and ReLU and max pooling layers between each layer and a Softmax activation function for the output layer.

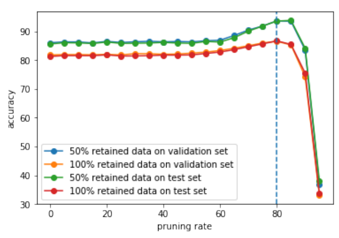
Experiment 1. Accuracy on 50% and 100% data with the lowest uncertainty



(a) Network with pretraining and iterative pruning



(b) Network without pretraining and with iterative pruning



(c) Network with pretraining and with one shot pruning

Figure B.5: Evolution of the accuracy with the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets for the three pruning techniques.

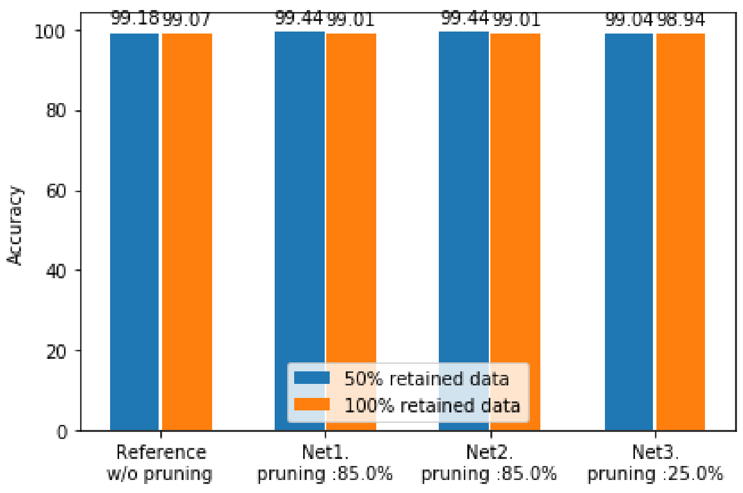
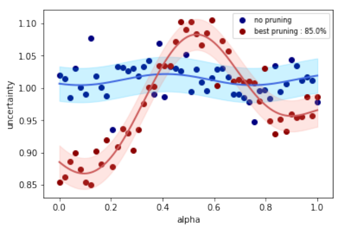
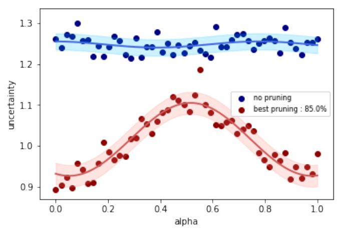


Figure B.6: Accuracy while keeping all data or the 50% with the lowest uncertainty on testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

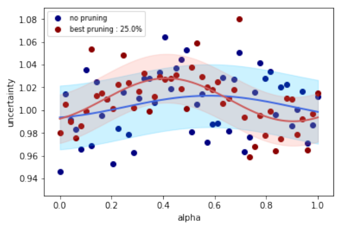
Experiment 2. Uncertainty estimates along with the interpolation between pairs of well-classified images.



(a) Network with pretraining and iterative pruning



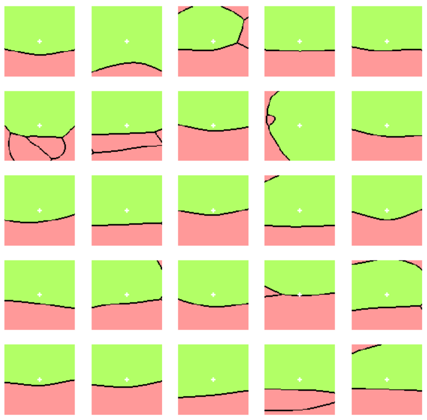
(b) Network without pretraining and with iterative pruning

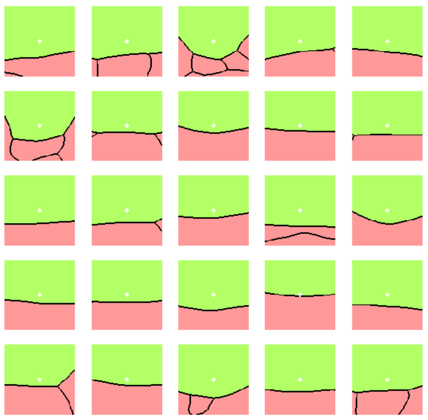


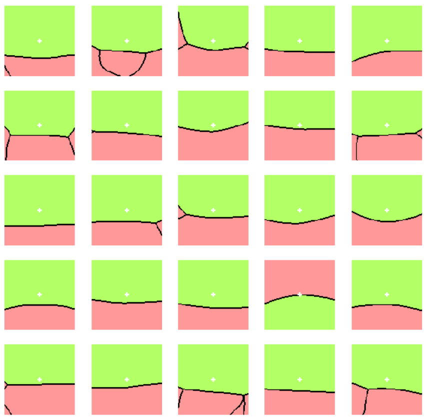
(c) Network with pretraining and with one shot pruning

Figure B.7: Evolution of the uncertainty estimate along the interpolation between 100 pairs of images taken in the test set.

Experiment 3. Visualisation of decision boundaries around 25 images.


(a) Reference network


(b) Network with pretraining and iterative pruning


(c) Network without pretraining and with iterative pruning

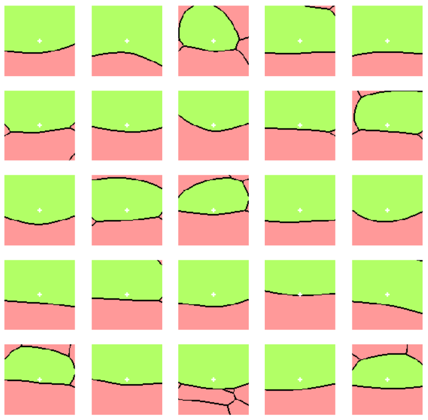
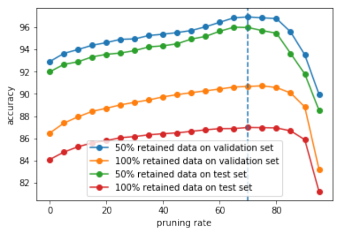

(d) Network with pretraining and with one shot pruning

Figure B.8: Decision boundaries for different networks.

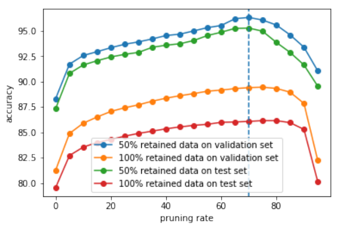
B.2 Fashion-MNIST

Fully connected neural networks. We use a fully connected neural network with almost 100'000 parameters. It is composed of two hidden layers with 128 and 64 neurons, a ReLU activation function between each layer and a Softmax activation function for the output layer.

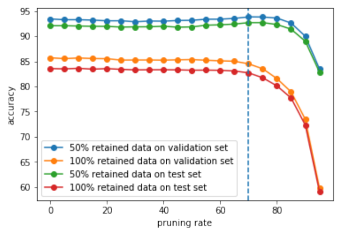
Experiment 1. Accuracy on 50% and 100% data with the lowest uncertainty



(a) Network with pretraining and iterative pruning



(b) Network without pretraining and with iterative pruning



(c) Network with pretraining and with one shot pruning

Figure B.9: Evolution of the accuracy with the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets for the three pruning techniques.

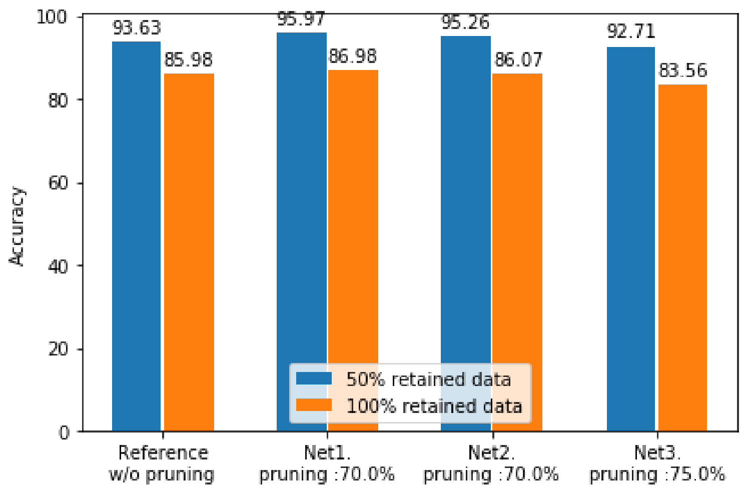
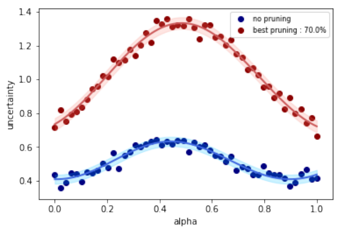
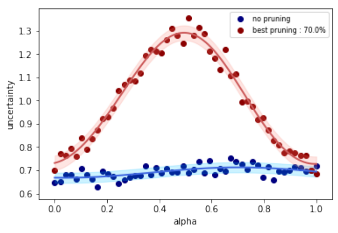


Figure B.10: Accuracy while keeping all data or the 50% with the lowest uncertainty on the testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

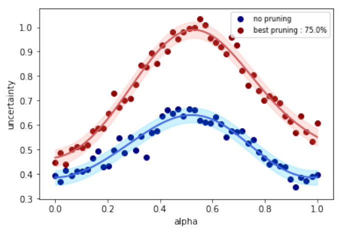
Experiment 2. Uncertainty estimates along with the interpolation between pairs of well-classified images.



(a) Network with pretraining and iterative pruning



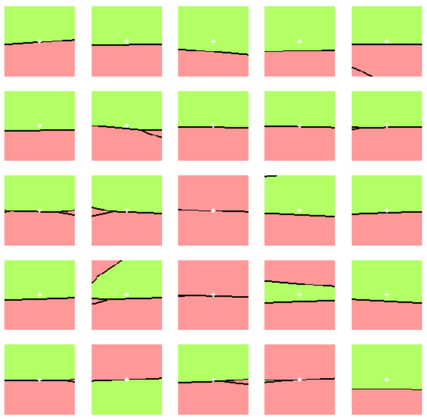
(b) Network without pretraining and with iterative pruning



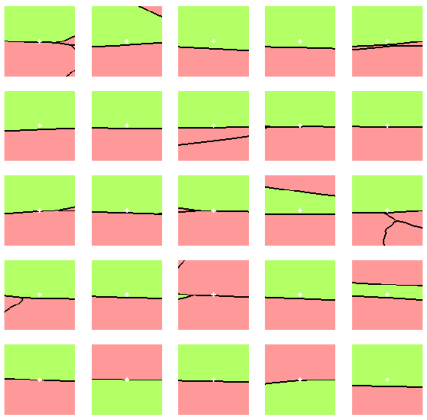
(c) Network with pretraining and with one shot pruning

Figure B.11: Evolution of the uncertainty estimate along the interpolation between 100 pairs of images taken in the test set.

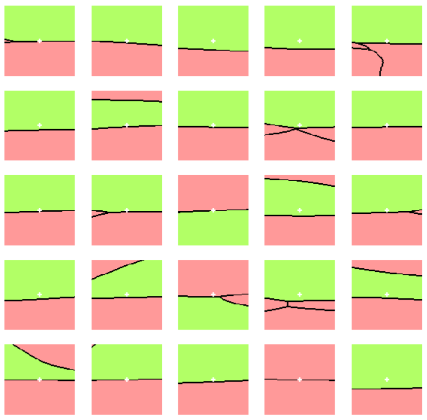
Experiment 3. Visualisation of decision boundaries around 25 images.



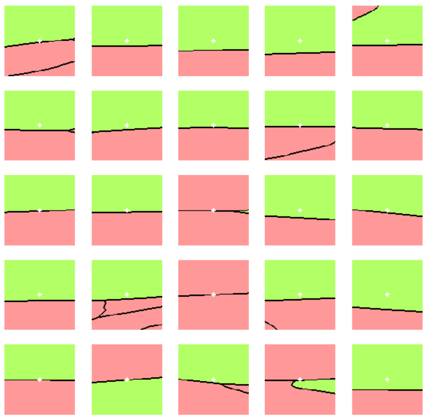
(a) Reference network



(b) Network with pretraining and iterative pruning



(c) Network without pretraining and with iterative pruning

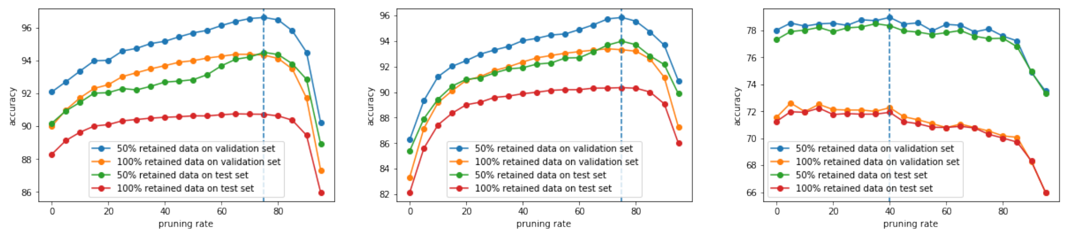


(d) Network with pretraining and with one shot pruning

Figure B.12: Decision boundaries for different networks.

Convolutional neural networks. We use a convolutional neural networks with almost 100'000 parameters. It is composed of two hidden layers with 64 channels kernel 5 by 5 and ReLU and max pooling layers between each layer and a Softmax activation function for the output layer.

Experiment 1. Accuracy on 50% and 100% data with the lowest uncertainty



(a) Network with pretraining and iterative pruning *(b) Network without pretraining and with iterative pruning* *(c) Network with pretraining and with one shot pruning*

Figure B.13: Evolution of the accuracy with the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets for the three pruning techniques.

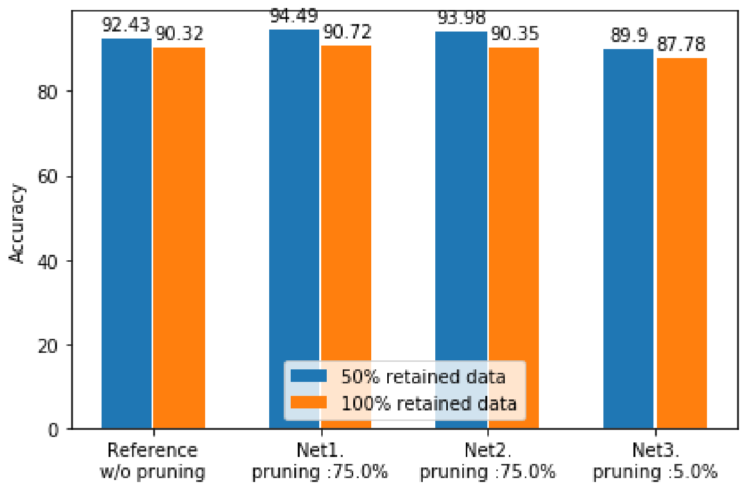
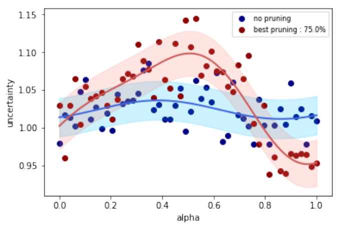
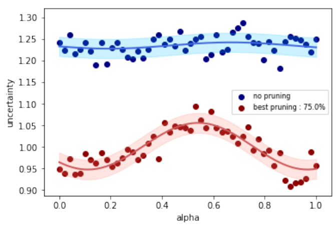


Figure B.14: Accuracy while keeping all data or the 50% with the lowest uncertainty on the testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

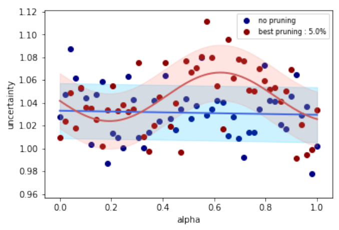
Experiment 2. Uncertainty estimates along with the interpolation between pairs of well-classified images.



(a) Network with pretraining and iterative pruning



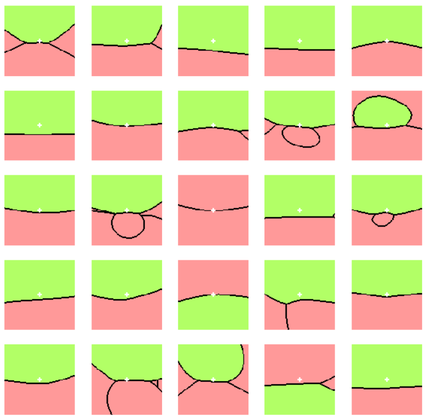
(b) Network without pretraining and with iterative pruning



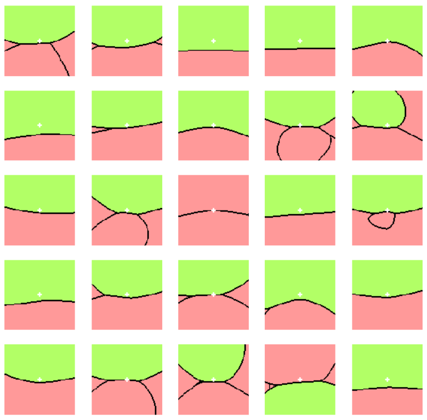
(c) Network with pretraining and with one shot pruning

Figure B.15: Evolution of the uncertainty estimate along the interpolation between 100 pairs of images taken in the test set.

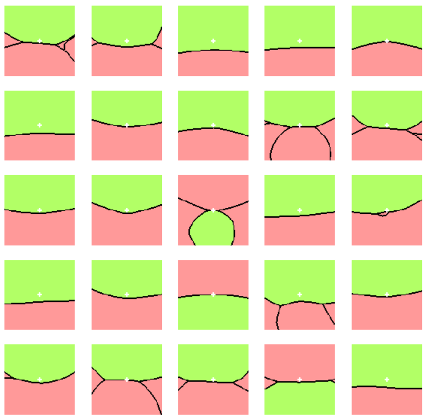
Experiment 3. Visualisation of decision boundaries around 25 images.



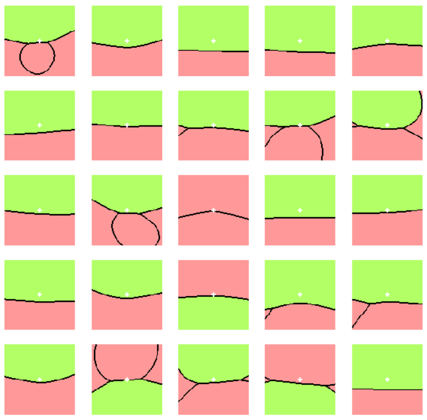
(a) Reference network



(b) Network with pretraining and iterative pruning



(c) Network without pretraining and with iterative pruning



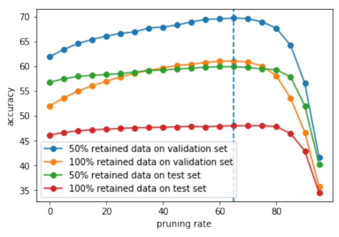
(d) Network with pretraining and with one shot pruning

Figure B.16: Decision boundaries for different networks.

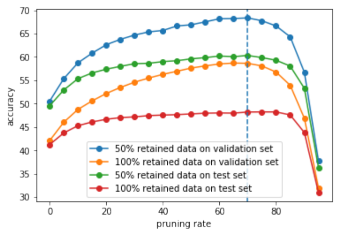
B.3 CIFAR-10

Fully connected neural networks. We use a fully connected neural network with almost 200'000 parameters. It is composed of two hidden layers with 64 and 64 neurons, a ReLU activation function between each layer and a Softmax activation function for the output layer.

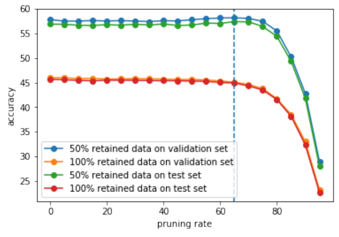
Experiment 1. Accuracy on 50% and 100% data with the lowest uncertainty



(a) Network with pretraining and iterative pruning



(b) Network without pretraining and with iterative pruning



(c) Network with pretraining and with one shot pruning

Figure B.17: Evolution of the accuracy with the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets for the three pruning techniques.

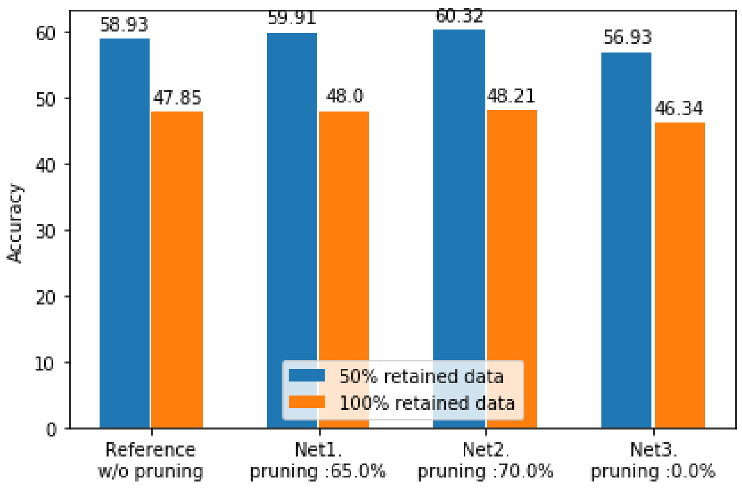
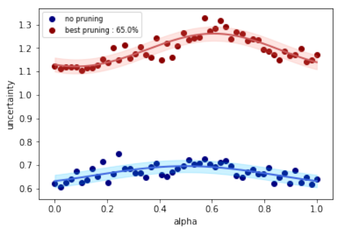
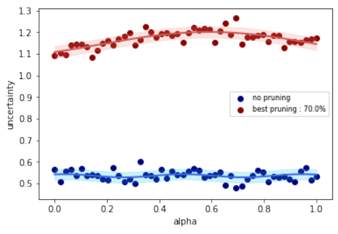


Figure B.18: Accuracy while keeping all data or the 50% with the lowest uncertainty on the testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

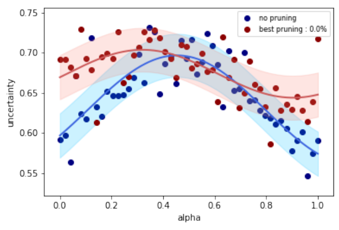
Experiment 2. Uncertainty estimates along with the interpolation between pairs of well-classified images.



(a) Network with pretraining and iterative pruning



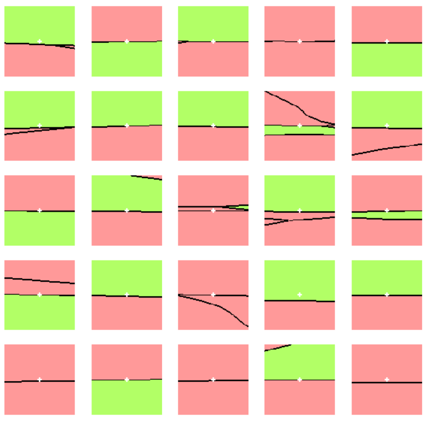
(b) Network without pretraining and with iterative pruning



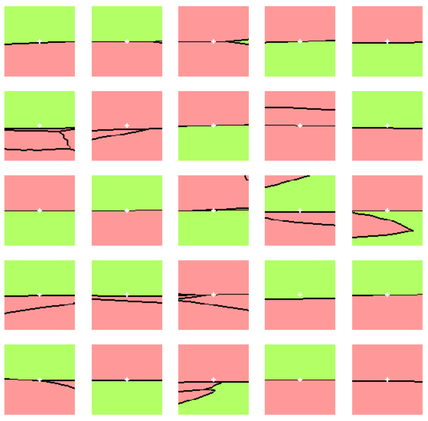
(c) Network with pretraining and with one shot pruning

Figure B.19: Evolution of the uncertainty estimate along the interpolation between 100 pairs of images taken in the test set.

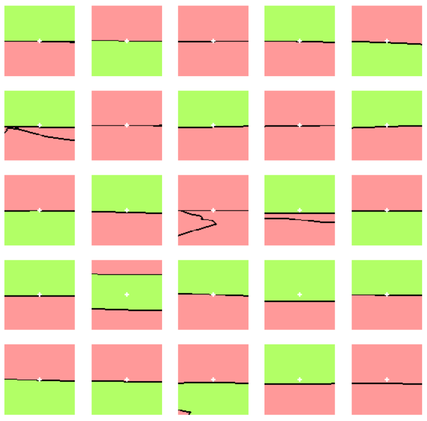
Experiment 3. Visualisation of decision boundaries around 25 images.



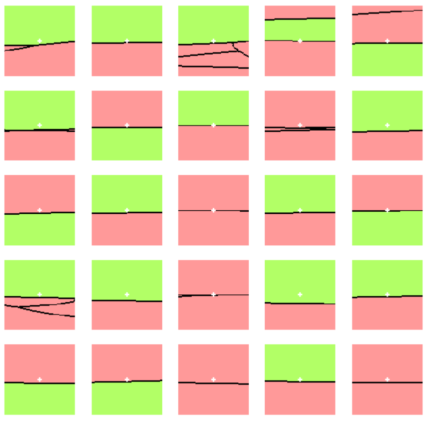
(a) Reference network



(b) Network with pretraining and iterative pruning



(c) Network without pretraining and with iterative pruning

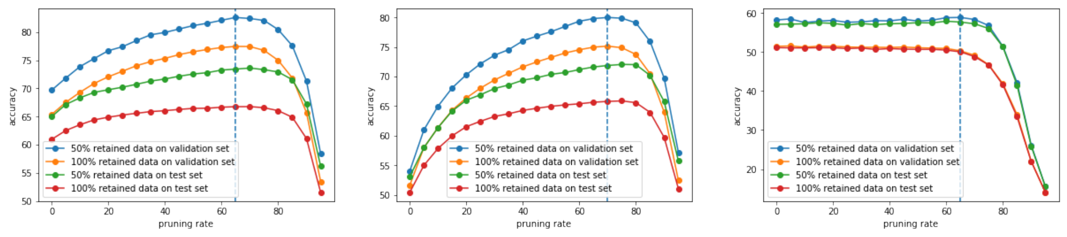


(d) Network with pretraining and with one shot pruning

Figure B.20: Decision boundaries for different networks.

Convolutional neural networks. We use a convolutional neural networks with almost 100'000 parameters. It is composed of two hidden layers with 64 channels kernel 5 by 5 and ReLU and max pooling layers between each layer and a Softmax activation function for the output layer.

Experiment 1. Accuracy on 50% and 100% data with the lowest uncertainty



(a) Network with pretraining and iterative pruning *(b) Network without pretraining and with iterative pruning* *(c) Network with pretraining and with one shot pruning*

Figure B.21: Evolution of the accuracy with the pruning rate while keeping all data or the 50% with the lowest uncertainty on both validation and test sets for the three pruning techniques.

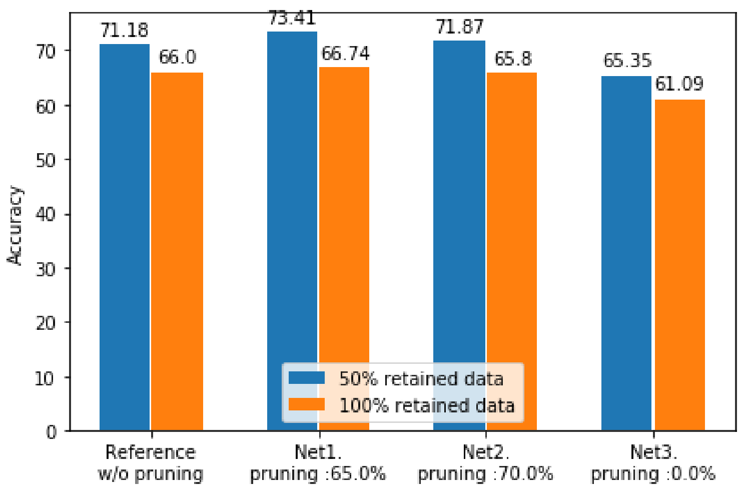
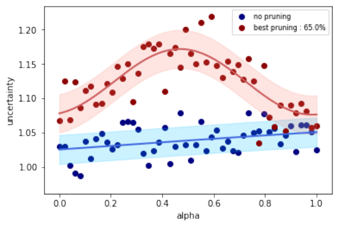
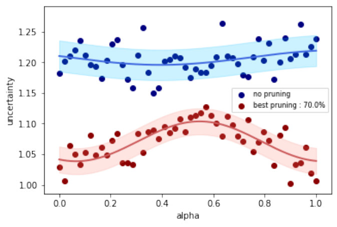


Figure B.22: Accuracy while keeping all data or the 50% with the lowest uncertainty on the testing set for the reference network and the network pruned with the best pruning rate based on performance on the validation set.

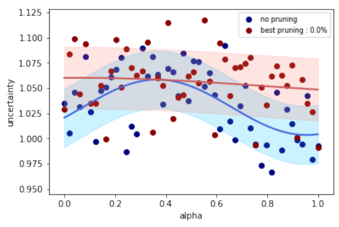
Experiment 2. Uncertainty estimates along with the interpolation between pairs of well-classified images.



(a) Network with pretraining and iterative pruning



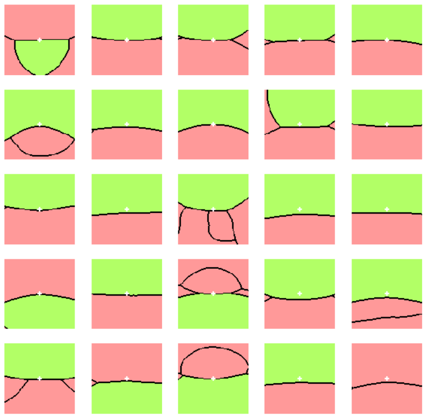
(b) Network without pretraining and with iterative pruning

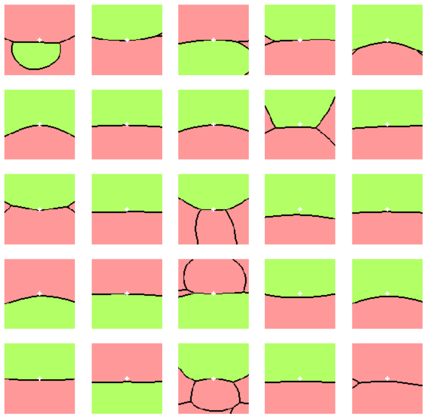


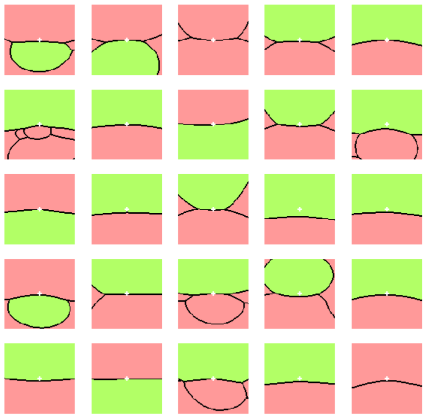
(c) Network with pretraining and with one shot pruning

Figure B.23: Evolution of the uncertainty estimate along the interpolation between 100 pairs of images taken in the test set.

Experiment 3. Visualisation of decision boundaries around 25 images.


(a) Reference network


(b) Network with pretraining and iterative pruning


(c) Network without pretraining and with iterative pruning

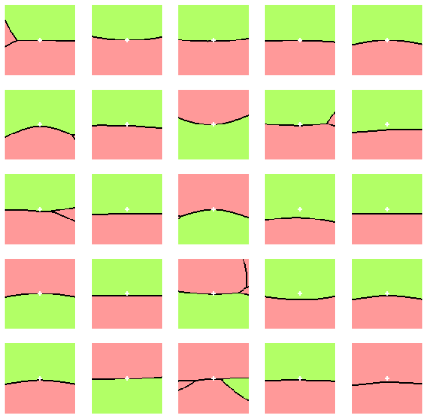

(d) Network with pretraining and with one shot pruning

Figure B.24: Decision boundaries for different networks.

Appendix C

Codes

All codes are provided in the ZIP file containing this report. They are also stored on the server lts2gdk0 in the directory /mnt/scratch/students/aguettaz. To run it, please use a similar directory structure as the one presented on figure C.1. The codes are organized as following: packages, functions, scripts. I tried to add comments to make codes understandable.

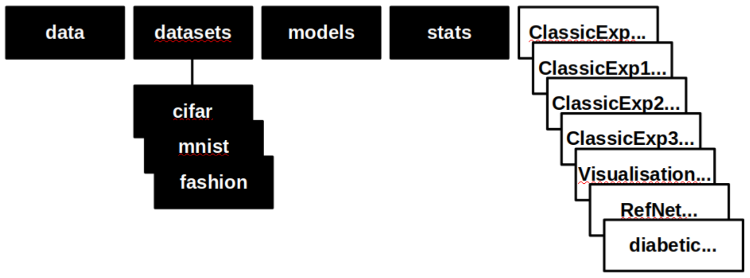


Figure C.1: Directory structure to launch the code

Classical experiments. Launched on ClassicExp's series, pruned models are stored in the directory models and are called according to the number of pretraining epochs, the pruning rate, the network type, and the dataset. The reference networks are built on RefNet. They are also stored on models. Some results are stored on the directory stats. Visualizations of the decision boundaries are computed on the notebook Visualisation but required to already have built some models.

Advanced experiments. Launched on diabetic retinopathy diagnosis notebook. It downloads another python script from GitHub to construct the dataset from Kaggle's one. Networks and results are stored on models and stats respectively.