université PARIS-SACLAY

NNT : 20XXSACLXXXX

Thèse de doctorat

Optimisation d'hyper-paramètres en apprentissage profond et apprentissage par transfert - Applications en imagerie médicale

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom ParisTech

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Traitement du signal et des images

Thèse présentée et soutenue à Paris, prévue le 15/01/2019, par

HADRIEN BERTRAND

Composition du Jury :

Hervé Delingette Directeur de recherche, INRIA Sophia-Antipolis	Rapporteur
Caroline Petitjean Maître de conférence, Université de Rouen	Rapporteuse
Pierrick Coupé Chargé de Recherche, Université de Bordeaux	Examinateur
Jamal Atif Professeur, Université Paris-Dauphine	Examinateur
Laurent Cohen Directeur de recherche, Université Paris-Dauphine	Examinateur
Isabelle Bloch Professeur, Télécom ParisTech	Directrice de thèse
Roberto Ardon Chercheur, Philips Research	Co-encadrant de thèse
Matthieu Perrot Chercheur, Philips Research	Co-encadrant de thèse

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Optimisation d'hyper-paramètres en apprentissage profond et apprentissage par transfert - Applications en imagerie médicale

Mots clés : Apprentissage profond, imagerie médicale, apprentissage par transfert, déformation de modèles, segmentation

Résumé : Ces dernières années, l'apprentissage profond a complètement changé le domaine de vision par ordinateur. Plus rapide, donnant de meilleurs résultats, et nécessitant une expertise moindre pour être utilisé que les méthodes classiques de vision par ordinateur, l'apprentissage profond est devenu omniprésent dans tous les problèmes d'imagerie, y compris l'imagerie médicale.
Au début de cette thèse, la construction de réseaux de neurones adaptés à des tâches spécifiques ne bénéficiait pas encore de suffisamment d'outils ni d'une compréhension approfondie. Afin de trouver automatiquement des réseaux de neurones adaptés à des tâches spécifiques, nous avons ainsi apporté des contributions à l'optimisation d'hyper-paramètres de réseaux de neurones. Cette thèse propose une comparaison de certaines méthodes d'optimisation, une amélioration en performance d'une de ces méthodes, l'optimisation bayésienne, et une nouvelle méthode d'optimisation d'hyper-paramètres basé sur la combinaison de deux méthodes existantes : l'optimisation bayésienne et hyperband.
Une fois équipés de ces outils, nous les avons utilisés pour des problèmes d'imagerie médicale : la classification de champs de vue en IRM, et la segmentation du rein en échographie 3D pour deux groupes de patients. Cette dernière tâche a nécessité le développement d'une nouvelle méthode d'apprentissage par transfert reposant sur la modification du réseau de neurones source par l'ajout de nouvelles couches de transformations géométrique et d'intensité.
En dernière partie, cette thèse revient vers les méthodes classiques de vision par ordinateur, et nous proposons un nouvel algorithme de segmentation qui combine les méthodes de déformations de modèles et l'apprentissage profond. Nous montrons comment utiliser un réseau de neurones pour prédire des transformations globales et locales sans accès aux vérités-terrains de ces transformations. Cette méthode est validé sur la tâche de la segmentation du rein en échographie 3D.

Title : Hyper-parameter optimization in deep learning and transfer learning - Applications to medical imaging

Keywords : Deep learning, medical imaging, transfer learning, template deformation, segmentation

Abstract : In the last few years, deep learning has changed irrevocably the field of computer vision. Faster, giving better results, and requiring a lower degree of expertise to use than traditional computer vision methods, deep learning has become ubiquitous in every imaging application. This includes medical imaging applications.
At the beginning of this thesis, there was still a strong lack of tools and understanding of how to build efficient neural networks for specific tasks. Thus this thesis first focused on the topic of hyper-parameter optimization for deep neural networks, i.e. methods for automatically finding efficient neural networks on specific tasks. The thesis includes a comparison of different methods, a performance improvement of one of these methods, Bayesian optimization, and the proposal of a new method of hyper-parameter optimization by combining two existing methods: Bayesian optimization and Hyperband.
From there, we used these methods for medical imaging applications such as the classification of field-of-view in MRI, and the segmentation of the kidney in 3D ultrasound images across two populations of patients. This last task required the development of a new transfer learning method based on the modification of the source network by adding new geometric and intensity transformation layers.
Finally this thesis loops back to older computer vision methods, and we propose a new segmentation algorithm combining template deformation and deep learning. We show how to use a neural network to predict global and local transformations without requiring the ground-truth of these transformations. The method is validated on the task of kidney segmentation in 3D US images.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

1.1.1 Medical Imaging

Automated methods have been developed to analyze medical images in order to facilitate the work of the clinicians. Tasks such as segmentation or localization of various body parts and organs, registration between frames of a temporal sequence or slices of a 3D image, or detection of tumors are well-suited to computer vision methods.

While computer vision methods developed for natural images can be reused, the specificity of medical images should be taken into account. Unlike natural images, medical images have no background, a more limited scope in the objects that are represented, no colors and an intensity that often has a precise meaning depending on the modality.

While these may seem to make the problem simpler at first glance, challenges in medical image analysis come in two categories. The first challenge is variability, either intra-subject (changes in the body with time, or during image acquisition as the patient breathes and moves) or inter-subject (in the shape, size and location of bones and organs). Variability also comes from external sources: patterns of noise specific to a machine, image resolution and contrasts depending on the image protocol, field-of-view depending on how the clinicians handle the probe ...

The second challenge comes from the difficulty of acquiring data and therefore the low amount of data available. This difficulty comes in many forms: the acquisition of images requires time and expertise, the sensitivity of the data adds ethical, administrative and bureaucratic overhead (the recent GDPR laws come to

mind), and sometimes the rarity of a disease makes it simply impossible to acquire large amounts of data.[1]

And acquiring the data is not enough! The data needs to be annotated, a task that often needs to be done by a clinical expert. Image segmentation is an important problem, but the manual creation of the necessary ground-truth segmentations is a time consuming activity that puts a strong limit on the size of medical images databases. A rough estimation: at an average of 5 minutes per image (a very optimistic time in many cases, in particular for 3D images), creating the ground-truth segmentations of a 100 images requires slightly over 8 hours of clinician time.

1.1.2 Deep Learning

These last few years, it has been impossible to talk about medical image analysis without talking about deep learning. The inescapable transformation brought with deep learning was made possible with the advent of cheap memory storage and computing power. At first glance, deep learning gives much better results than traditional computer vision algorithms, while often being faster. From its first big success in the 2012 ImageNet competition won by Krizhevsky, Sutskever, and Hinton (2012), deep learning has had a string of successes that now makes it ubiquitous in computer vision, including medical imaging.

This change brought with it its own set of challenges. The huge amount of resources invested into the field results in a deluge of publications where it is difficult to keep up-to-date and separate the noise from the actual progress. New algorithms and technologies go from research to production to widely used so fast, the field has completely changed in the duration of this thesis.

To give some perspectives on the progress, at the start of this thesis in early 2016:

- Tensorflow (Martín Abadi et al. (2015)) was just released and a nightmare to install - now it works everywhere from desktop to smartphone and has been cited in over 5000 papers.

- The original GAN paper by Goodfellow *et al* was published mid 2014 (Goodfellow, Pouget-Abadie, et al. (2014)). Early 2016, GANs had the reputation of being incredibly difficult to train and unusable outside of toy datasets.

[1] Initiatives are underway to create big and accessible medical images databases. See for examples the NIH Data Sharing Repositories or the Kaggle Healthcare datasets.

Three years and 5000 citations later, GANs have been applied to a wide range of domains, including medical imaging.

- The now omnipresent U-Net architecture had just made its debut a few months earlier at MICCAI 2015 (Ronneberger, Fischer, and Brox (2015)).

1.2 Contributions and Outline

This thesis is at the cross-road of medical image analysis and deep learning. It first started as an exploration of how to use deep learning on medical imaging problems. The first roadblock encountered was the construction of neural networks specific to a problem. Their was a lack of understanding of the effect of each component of the network on the task to be solved (this is still the case). How many convolutional layers are needed to solve this task? Is it better to have more filters or bigger filters? What is the best batch size? The lack of answers to those questions led us to the topic of hyper-parameter optimization; if we cannot lean on theoretical foundations to build our models, then at least we can automate the search of the best model for a given task and have an empirical answer.

Once equipped with hyper-parameter optimization tools, we turned to applications, first the classification of field-of-view in MR images, then the segmentation of the kidney in 3D ultrasound images. This last problem was examined in a transfer learning setting and led us to the development of a new transfer learning method.

In the final part of this thesis we returned to older computer vision methods, notably template deformation methods, and proposed a new method to combine them with deep learning.

This thesis is structured in three parts, each starting with a review of the relevant literature.

- **Chapter 2** discusses the topic of hyper-parameter optimization in the context of deep learning. We focus on three methods: random search, Bayesian optimization and Hyperband. The chapter includes (1) a performance improvement of Bayesian optimization by using an incremental Cholesky decomposition; (2) a theoretical bound on the performance of random search and a comparison of random search and Bayesian optimization in a practical setting; (3) a new hyper-parameter optimization method combining Hyperband and Bayesian optimization, published as Bertrand, Ardon, et al. (2017);

(4) an application of Bayesian optimization to solve a classification problem of MRI field-of-view, published as Bertrand, Perrot, et al. (2017).

- **Chapter 3** introduces a new transfer learning method in order to solve the task of segmenting the kidney in 3D ultrasound images across two populations: healthy adults and sick children. The challenge comes from the high variability of the children images and the low amount of images available, making transfer learning methods such as fine-tuning insufficient. Our method modifies the source network by adding layers to predict geometric and intensity transformations that are applied to the input image. The method was filled as a patent and is currently under review.

- **Chapter 4** presents a segmentation method that combines deep learning with template deformation. Building on top of the *implicit template deformation* framework, a neural network is used to predict the parameters of a global and a local transformation, which are applied to a template to segment a target. The method requires only pairs of image and ground-truth segmentation to work, the ground-truth transformations are not required. The method is tested on the task of kidney segmentation in 3D US images.

- **Chapter 5** summarizes our conclusions and discusses possible future works.

2 Hyper-parameter Optimization of Neural Networks

Abstract

This chapter introduces the problem of hyper-parameter optimization for neural networks, which is the problem of automatically finding the optimal architecture and training setting for a particular task. Section 2.1 presents the problem and the existing approaches while Section 2.2 explains in depth the method of Bayesian optimization which is used for the rest of the chapter. A performance improvement of this method is given in Section 2.3. In Section 2.4, we compare the performance of random search and Bayesian optimization on a toy problem and give theoretical bounds on the performance of random search. We propose a new method in Section 2.5, which combines Bayesian optimization with another method, Hyperband. Finally, we apply Bayesian optimization to the practical problem of MRI field-of-view classification in Section 2.6.

Contents

2.1 Defining the Problem

The problem of hyper-parameter optimization appears when a model is governed by hyper-parameters, i.e. parameters that are not learned by the model but must be chosen by the user. Automated methods for tuning them become worthwhile when hyper-parameters are numerous and difficult to manually tune due to a lack of understanding of their effects. The problem gets worse when the hyper-parameters are not independent and it becomes necessary to tune them at the same time. The most intuitive solution is to test all possible combinations by discretizing the hyper-parameter space and choosing an order of evaluation, a method called *grid search*. This method does not scale because the number of combinations grows exponentially with the number of hyper-parameters, making this approach usually unusable for neural networks.

In practice, as it is usually impossible to prove the optimality of a solution without testing all solutions, the accepted solution is the best found in the budget allocated by the user to the search.

Even though this problem appears in all kinds of situations, we are interested here in the optimization of deep learning models, as they have many hyper-parameters and we lack the understanding and the theoretical tools to tune them. The hyper-parameters fall in two categories: the ones that affect the architecture of the network (such as the number or type of layers) and the ones that affect the training of the network (such as the learning rate or the batch size).

In the case of deep learning, evaluating a combination (a specific network) is costly in time and computing resources. Moreover we have little understanding of the influence of each hyper-parameter on the performance of the model, resulting in very large boundaries and a much bigger space than needed. The situation is even worse since combinations near the boundaries of the hyper-parameter space can build a network too big to fit in the available memory, requiring to have a way to handle evaluation failures.

While it is tempting to simply return an extremely bad performance, this causes problems to methods assuming some structure in the hyper-parameter space. For example, a continuous hyper-parameter that gives a smooth output will suddenly have a discontinuity, and a method assuming smoothness (such as Bayesian optimization) might not work anymore. In practice, there are method-dependent solutions to this problem.

2.1.1 Notations

In this chapter we refer to a *model* as a neural network, even though the black-box methods presented in Section 2.1.2 work for other machine learning models. The *parameters* of a model are the weights of the network which are learned during training. The *hyper-parameters* are the parameters governing the architecture of the networks (such as the number of layers or the type of layers) and the ones governing the training phase (such as the learning rate or the batch size).

A *hyper-parameter space* X is a hypercube where each dimension is a hyper-parameter and the boundaries of the hypercube are the boundaries of each hyper-parameter. Each point in the hyper-parameter space is referred to as a *combination* $\mathrm{x} \in \mathrm{X}$. To each combination is associated a value y corresponding to the performance metric of the underlying neural network on the task it is trained to solve. We name f a function taking as input a combination x, build and train the underlying model and output its performance $f(\mathrm{x})$.

Concretely in the case of deep learning, f is the process that build and train a neural network specified by a set of hyper-parameters x, and returns the loss of the model which serves as the performance metric.

2.1.2 Black-box optimization

From the notation of Section 2.1.1, the goal of hyper-parameter optimization is to find the combination x_* minimizing the performance y (typically the performance is a loss function, which we want to minimize):

$$\mathrm{x}_* = \underset{\mathrm{x} \in \mathrm{X}}{\operatorname{argmin}} f(\mathrm{x}) \tag{2.1}$$

This is the problem of hyper-parameter optimization viewed as black-box optimization. Methods in this category are independent from the model and could be used for any kind of mathematical function. However the hyper-parameters we are interested in are of a varied nature (continuous, discrete, categorical), limiting us to derivative-free optimization methods. This section is not a complete review of derivative-free algorithms, but an overview of the popular algorithms used specifically for hyper-parameter optimization.

2.1.2.1 Grid search

The most basic method is usually called *grid search*. Very easy to implement, it simply tests every possible combination (typically with uniform sampling for the continuous hyper-parameters). With only a handful of hyper-parameters to optimize and with a function f fast to evaluate, it can be advisable to use as a first step to get sensible boundaries on each hyper-parameter, i.e. by testing a handful of values over a very wide range to find a smaller but relevant range.

But in the context of deep learning, hyper-parameters are too numerous, meaning there are too many combinations to evaluate, and each evaluation is costly. Limiting oneself when choosing the hyper-parameters still leaves us with at least 5 or 6 hyper-parameters but it is easy to end up with over 30 hyper-parameters. Assuming an average of 4 values per hyper-parameters, this implies $4^5 = 1024$ combinations for a space of 5 hyper-parameters or $4^{30} \simeq 10^{18}$ combinations with 30 hyper-parameters. Grid search does not scale well, which makes it unsuitable for deep learning.

Even for reasonably sized hyper-parameter space, a typical implementation in nested for-loops goes from one corner of the hyper-parameter space to the opposite corner in fixed order. It is unlikely that the corner the search starts from happens to be an area filled with good models, since combinations in corners are extreme values of hyper-parameters and build atypical neural networks.

Grid search can still be used with a wide enough sampling of the hyper-parameters to get an idea on where the interesting models are located and refine the boundaries, but even for that other methods are preferable.

2.1.2.2 Random search

One step above grid search is *random search*. A big limitation of grid search is that the order it goes through the hyper-parameter space is very dependent on the implementation and it always selects the same limited set of values for each hyper-parameter. Random search instead draws the value of each hyper-parameter from a uniform distribution, allowing for a much wider range of explored values.

Figure 2.1 illustrates this. Given a hyper-parameter space of two continuous hyper-parameters, and at equal number of evaluated combinations, random search finds a better solution. It works because hyper-parameters are not equally relevant. In practice, for deep learning, only a few have a high impact on the performance (Bergstra and Bengio (2012)).[1] In the figure, grid search only tested three values

[1] We do not know in advance which hyper-parameters have a high impact, otherwise we would

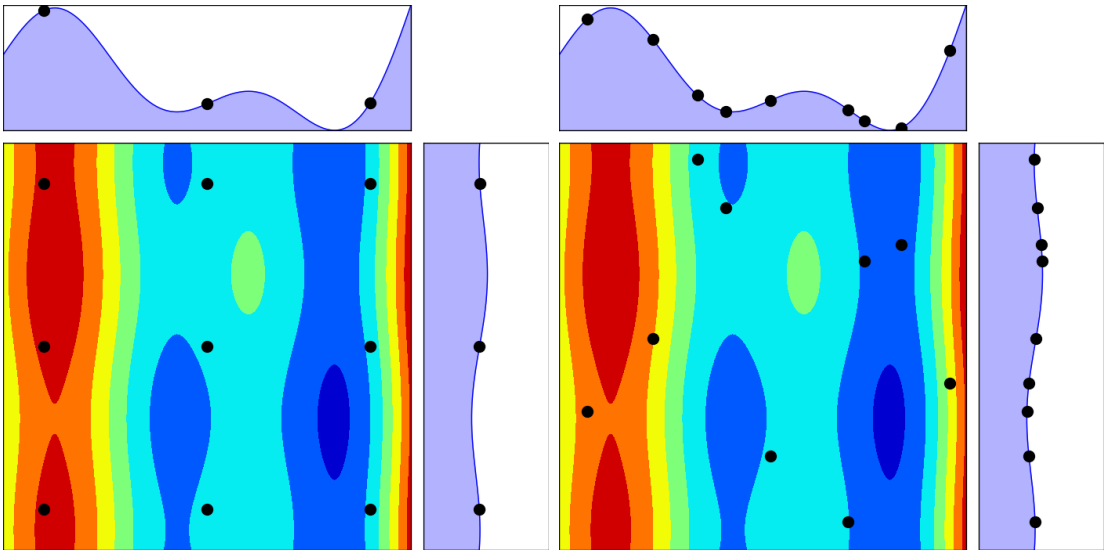


Figure 2.1: Loss function for a two dimensional space. (Left) Grid search. (Right) Random search. 9 combinations were tested for each method, but grid search tested only 3 values for each hyper-parameters while random search tested 9, resulting in a better solution.

of each hyper-parameters while random search tested nine for the same cost!

In terms of implementation cost, random search requires only the ability to draw uniformly from an interval or a list, giving it the same computational cost than grid search. The increased implementation cost is largely compensated by the gain in performance.

2.1.2.3 Bayesian optimization

Going further requires some assumption about the structure of the hyper-parameter space. Namely, similar values for an hyper-parameter results in similar performance, i.e. some weak notion of continuity. If there is some structure, we can exploit it.

Bayesian optimization (Bergstra, Bardenet, et al. (2011)), which we present in details in Section 2.2, applies this idea by modeling f with as few evaluations as possible while searching for the best combination. It comes with a non-trivial implementation cost, though packages are available. In Section 2.4 we compare the performance of random search and Bayesian optimization on a practical task.

tune only high impact hyper-parameters.

2.1.3 Evolutionary algorithms

The first application of evolutionary algorithms to neural networks was to evolve the weights of a fixed architecture (Miller, Todd, and Hegde (1989)). A few years later, Braun and Weisbrod (1993) and Angeline, Saunders, and Pollack (1994) realized it was more efficient to evolve simultaneously the weights and the topology of the network. Further refinements of these approaches led to the *NeuroEvolution of Augmenting Topologies* (NEAT) algorithm (Stanley and Miikkulainen (2002)).

NEAT uses three kinds of mutations (modifying a weight, adding a connection between two nodes and adding a node in the middle of a connection) and one kind of recombination based on fitness sharing. Many variants of this algorithm have been proposed, notably HyperNEAT (Stanley, D'Ambrosio, and Gauci (2009)) which only evolves the topology and learns the weights by back-propagation.

The main drawback of NEAT and its variants is their inability to scale and evolve the large networks typical of modern deep learning. Real et al. (2017) and Miikkulainen et al. (2017) developed approaches able to explore extremely diverse and atypical network structures and find architectures with state-of-the-art performance on computer vision tasks such as CIFAR-10.

The drawback of those methods however is their excessive computational cost, requiring thousands of iterations before starting to build good models. Coupled with their novelty, those methods were out of our reach, while the lack of scaling was too important in the older methods for us to use them in this thesis.

2.1.4 Reinforcement learning

Recent advances in reinforcement learning have made possible its use to design efficient neural networks. The idea is to train an agent called the controller which builds neural networks for a specific task. Baker et al. (2017) developed a controller that chooses each layer of the network sequentially and is trained using Q-learning. B. Zoph and Le (2017) created a string representation of neural networks and their controller is a RNN outputting valid strings. In both cases the created networks must be fully trained to be evaluated and the controller takes thousands of iterations to converge, making those approaches extremely costly.

To address this problem, Barret Zoph et al. (2017) proposed testing on a smaller dataset as an approximation of the performance on the true dataset. Another suggestion is to train only partially each network (Li et al. (2017), Zela et al. (2018)), allowing longer training time as the search is refined.

While those approaches are very promising, they are too recent and costly for this thesis.

2.1.5 Other approaches

We finish our tour with a few methods taking radically different approaches. Hyperband (Li et al. (2017)) considers the question as a multi-armed bandit problem, where each arm is a combination and we have a finite amount of training budget to allocate to each arm. The idea is to train many models partially, and take the decision to continue training every few epochs based on the current performance. Since we chose to explore this approach, we present it in more details in Section 2.5.1.

Hazan, Klivans, and Yuan (2018) proposed an approach applying compressed sensing to hyper-parameters optimization called Harmonica. Like Bayesian optimization, they aim to learn the structure of the hyper-parameter space, but using a sparse polynomial.

Finally, Domhan, Springenberg, and Hutter (2015) suggested predicting the learning curve of a model to decide whether to continue the training. The prediction is done using a portfolio of parametric models.

2.1.6 Synthesis

We have presented a variety of methods that have been used to optimize the hyper-parameters of neural networks. We summarize their advantages and differences in Table 2.1 according to several important criteria.

By black-box we ask whether the method makes any assumption about the model being optimized. Those are the methods that can be reused without modifications to optimize other machine learning models. The second criterion is if the method requires some structure in the hyper-parameter space to work. Would the method still work if we randomize the mapping from combination to performance?

The next two criteria are about the kind of hyper-parameters the method can work with. By conditional hyper-parameter we mean hyper-parameters whose relevance depends on the value of another hyper-parameter, for example the number of filters of a convolutional layer is only relevant if the layer exists. The second class of hyper-parameters are the one altering the training of the model instead of its topology, such as the learning rate or the batch size.

The last three criteria are subjective but relevant when choosing a method to

	Grid Search	Random Search	Bayesian Optimization - GP	Bayesian Optimization - Tree	Evolutionary Algorithms	Reinforcement Learning	Extrapolation of Learning Curves	Hyperband	Hannonica
Black-box?	✓	✓	✓	✓					✓
Does not require structure in X?	✓	✓						✓	
Conditional hyper-parameters?				✓	✓	✓			
Training method hyper-parameters?	✓	✓	✓	✓			✓[1]	✓[1]	✓
Easy to parallelize?		✓			✓			✓	
Method complexity?	Low	Low	Mid	Mid	High	High	Mid	Mid	High
Budget required?	Mid	Mid	Low	Low	High[2]	High[2]	Low	Mid	Low

Table 2.1: Comparison of different hyper-parameter optimization methods according to various criteria explained in Section 2.1.6. (1) Except for the number of epochs or training time which is controlled by the method. (2) This is quickly being reduced.

use. If the method allows training multiple models at the same time (therefore allowing using many GPUs) without major changes, we consider it easily parallelizable. The difficulty of understanding and implementing the method is evaluated on a coarse low to high scale. We use the same scale for the budget required to find some of the best models of a hyper-parameter space.

2.1.7 Conclusion

The first methods used for the optimization of hyper-parameters in neural networks were, without surprise, existing hyper-parameters optimization methods treating the model as black-box and which were developed for other models. While still competitive, they are limited in the architectures of networks they can explore, often working at a layer-level.

Modern approaches based on evolutionary algorithms or reinforcement learning focus on much more fine-grained architecture choices, at the unit-level. This shift from optimizing hyper-parameters to building architecture is marked notably by the problem increasingly being called Neural Architecture Search. While those methods are too resource intensive to be used in most cases, this is changing very rapidly (from B. Zoph and Le (2017) who used 800 GPUs for a month to Pham et al. (2018) who used a single GPU for less than a day).

But in the context of this thesis we worked with what was the most relevant at the time in 2016, namely Bayesian optimization, and to some extent, Hyperband.

2.2 Bayesian Optimization

Bayesian Optimization is a method for optimizing the parameters of a black-box that is costly to evaluate. In deep learning the black-box is a neural network and the parameters are the hyper-parameters of the network. Evaluating the network corresponds to training it and computing its performance on the validation set. A recent and general review of the topic can be found in Shahriari et al. (2016) where it is treated as an optimization method, while Snoek, Larochelle, and Adams (2012) review the topic in the context of optimizing the hyper-parameters of machine learning models.

There are two components in Bayesian optimization methods. The first component is a probabilistic model of the loss function, i.e. a function that takes the values of the hyper-parameters as input and estimate the value of the loss

the corresponding neural network would have. Gaussian processes are the typical choice and are presented in Section 2.2.1. The second component, called the acquisition function, samples the model of the loss function to select the next set of hyper-parameters to evaluate. Common acquisition functions are presented in Section 2.2.2.

2.2.1 Gaussian processes

A Gaussian process is a supervised learning model mainly used for regression problems. It is a distribution over functions, i.e. from a set of data points, the Gaussian process gives possible functions that fit those points, weighted by their likelihood. The shape and properties of possible functions are defined by a covariance function. When predicting the value of an unseen point, the Gaussian process returns a Normal distribution, with the variance being an estimation of the uncertainty of the model at this point. Predicting multiple points will result in a joint Gaussian distribution. A comprehensive review of the topic can be found in Rasmussen and Williams (2005)

2.2.1.1 Definitions

Following the notation of Section 2.1.1, we write the Gaussian process as:

$$\mathrm{y}(\mathrm{x}) \sim \mathcal{GP}\left(m(\mathrm{x}), k(\mathrm{x}, \mathrm{x})\right) \tag{2.2}$$

$m(\mathrm{x})$ is the *mean function* and $k(\mathrm{x}, \mathrm{x}')$ is the *covariance function* which specifies the covariance between pair of data points. The mean function is set to 0 for simplicity. In practice this is ensured by removing the mean of the predicted values from the dataset. The covariance function is used to build the covariance matrix of a set X of N data points as:

$$K(\mathrm{X}, \mathrm{X}) = \begin{pmatrix} k(\mathrm{x}_1, \mathrm{x}_1) & k(\mathrm{x}_1, \mathrm{x}_2) & \cdots & k(\mathrm{x}_1, \mathrm{x}_\mathrm{N}) \\ k(\mathrm{x}_2, \mathrm{x}_1) & k(\mathrm{x}_2, \mathrm{x}_2) & \cdots & k(\mathrm{x}_2, \mathrm{x}_\mathrm{N}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathrm{x}_\mathrm{N}, \mathrm{x}_1) & k(\mathrm{x}_\mathrm{N}, \mathrm{x}_2) & \cdots & k(\mathrm{x}_\mathrm{N}, \mathrm{x}_\mathrm{N}) \end{pmatrix} \tag{2.3}$$

This matrix is all that is needed to draw samples from the distribution. We pick a set X_* of points, build the covariance matrix $K(\mathrm{X}_*, \mathrm{X}_*)$, then generate samples

from this Gaussian distribution:

$$\mathrm{y}_* \sim \mathcal{N}\left(0, K(\mathrm{X}_*, \mathrm{X}_*)\right) \tag{2.4}$$

Some such samples are shown in Figure 2.2. Since no data points were used, this corresponds to an unfitted Gaussian process.

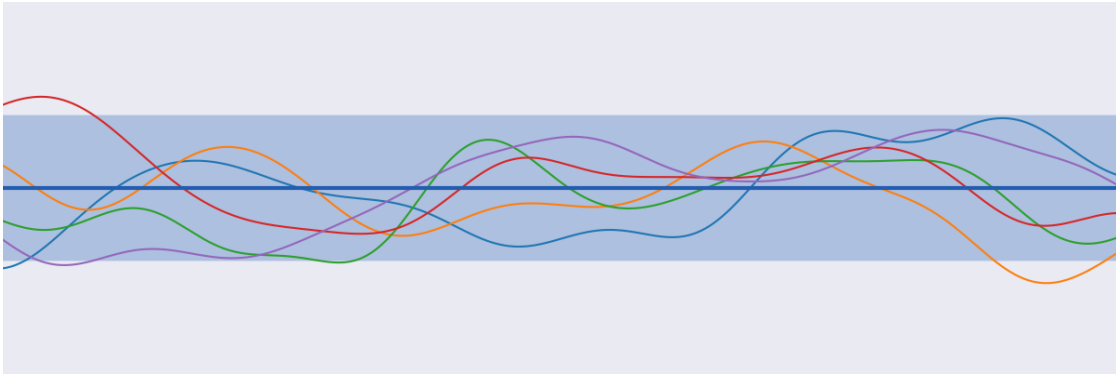


Figure 2.2: Gaussian process prior using a squared exponential kernel. The thin colored lines are samples drawn from the Gaussian process. The thick blue line represents the mean of the distribution and the blue area around this line is the 95% prediction interval, i.e. all drawn samples will be in this interval with a probability of 95% (for a Normal distribution this is equal to 1.96σ)

In probabilistic terms, the unfitted Gaussian process corresponds to the *prior* probability $p\left(\mathrm{y}\right)$. Given a set of observed points (X, y) and a set of points we want to predict $(\mathrm{X}_*, \mathrm{y}_*)$, the prior corresponds to $p\left(\mathrm{y}_*|\mathrm{X}_*, \theta\right)$, where θ denote the set of hyper-parameters of the covariance function. We are interested in the *posterior* probability $p\left(\mathrm{y}_*|\mathrm{X}_*, \mathrm{X}, \mathrm{y}, \theta\right)$ i.e. the distribution of the new points conditioned on the points we have already observed. From probability theory we know that the posterior is:

$$p\left(\mathrm{y}_*|\mathrm{X}_*, \mathrm{X}, \mathrm{y}, \theta\right) = \frac{p\left(\mathrm{y}, \mathrm{y}_*|\mathrm{X}, \mathrm{X}_*, \theta\right)}{p\left(\mathrm{y}|\mathrm{X}, \theta\right)} \tag{2.5}$$

The numerator is called the *joint distribution* and the denominator is the *marginal likelihood*.

2.2.1.2 Inference

Training a Gaussian process simply means pre-computing $K(\mathrm{X}, \mathrm{X})$, i.e. the covariance between the data points. At inference, we compute the covariance $K(\mathrm{X}_*, \mathrm{X}_*)$

between the points we want to predict, and the covariance between the data points and the points we want to predict $K(\mathrm{X}, \mathrm{X}_*)$. Since the covariance matrix is by definition symmetrical, $K(\mathrm{X}_*, \mathrm{X}) = K(\mathrm{X}, \mathrm{X}_*)^T$. For notational simplicity we denote them K, K_* or K_*^T and K_{**}.

This results in the joint distribution of Equation 2.5:

$$\begin{bmatrix} \mathrm{y} \\ \mathrm{y}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(\mathrm{X}, \mathrm{X}) & K(\mathrm{X}, \mathrm{X}_*) \\ K(\mathrm{X}_*, \mathrm{X}) & K(\mathrm{X}_*, \mathrm{X}_*) \end{bmatrix}\right) = \mathcal{N}\left(0, \begin{bmatrix} K & K_*^T \\ K_* & K_{**} \end{bmatrix}\right) \tag{2.6}$$

However the joint distribution generate functions that do not match the observed data. The posterior is obtained by conditioning the joint distribution to the observations, which are expressed by the marginal likelihood. Because every term involved is a Gaussian distribution, the posterior can be derived as follows:

$$p\left(\mathrm{y}_* | \mathrm{X}_*, \mathrm{X}, \mathrm{y}, \theta\right) = \mathcal{N}\left(K_* K^{-1} y, K_{**} - K_* K^{-1} K_*^T\right) \tag{2.7}$$

This is the equation to compute when predicting new points.

Figure 2.3 shows how samples from this distribution look like on a one dimensional problem. Each sample must go through every observed point, and the closer the points are, the less freedom the samples have to change. Outside of the range of observed points, the distribution quickly reverts to its prior.

2.2.1.3 Kernels

The most common kernel is the squared exponential kernel:

$$k(\mathrm{x}, \mathrm{x}') = \sigma^2 \exp\left(-\frac{||\mathrm{x} - \mathrm{x}'||_2^2}{2l^2}\right) \tag{2.8}$$

With this kernel, the influence of a point on the value of another point decays exponentially with their relative distance. This implies that the Gaussian process quickly reverts to its prior in areas without observed points.

This kernel has two hyper-parameters $\theta = \{\sigma^2, l\}$. σ^2 controls the scale of the predicted output and l is a vector of same dimensionality as x called the characteristic length-scale which measures how much a change along each dimension affects the output. A low value means that a small change in the input results in a big change in the output, as shown in Figure 2.4.

The squared exponential kernel is a special case of the Matérn family of covari-

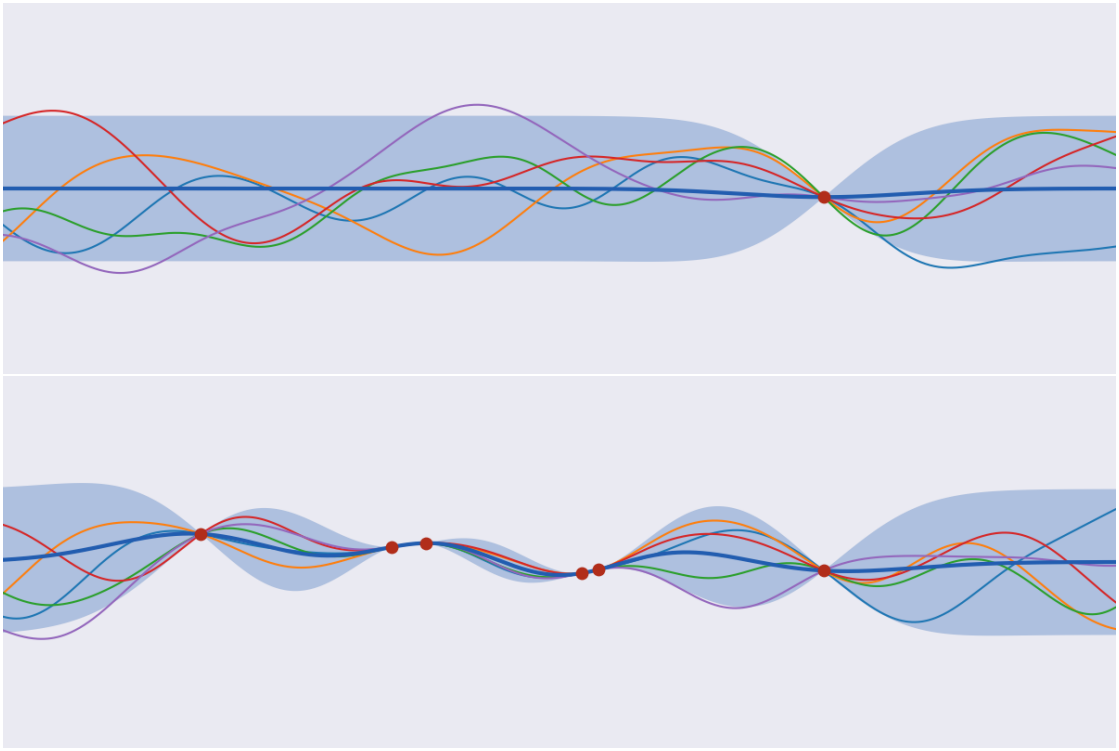


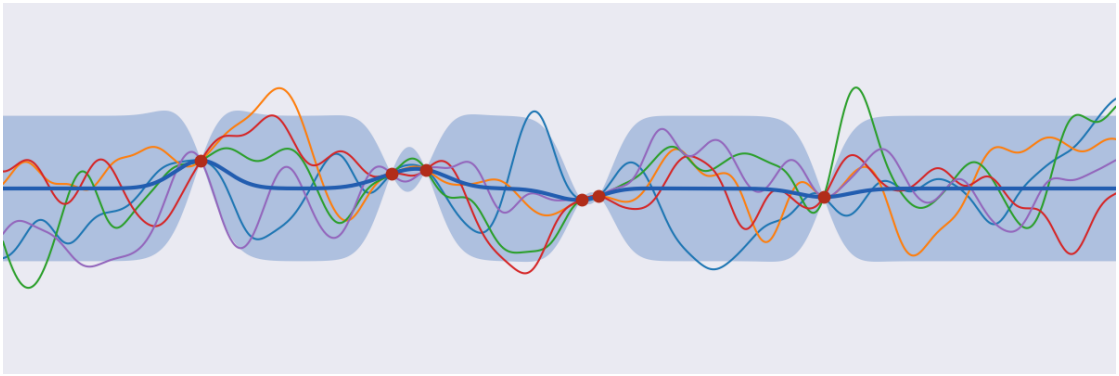
Figure 2.3: Gaussian process posterior after fitting one point on top, six on the bottom. All the samples pass through those points and the variance is lower close to them.

ance functions which are, noting $r = ||\mathrm{x} - \mathrm{x}'||_2$ defined as follows:

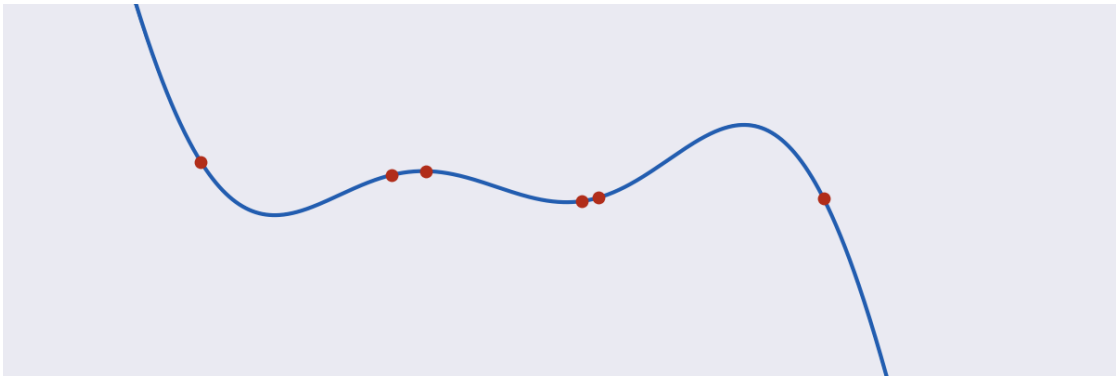
$$k_\nu(r) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{r}{l}\right)^\nu B_\nu \left(\sqrt{2\nu}\frac{r}{l}\right) \tag{2.9}$$

Γ is the gamma function and B_ν is the modified Bessel function of the second kind. The hyper-parameters σ^2 and l are the same as for the squared exponential kernel and ν is a measure of how smooth the function is. $\nu = 1/2$ results in a very rough function while $\nu \to \infty$ is the squared exponential kernel.

The samples from the squared exponential kernel are very smooth, and are in fact infinitely differentiable. This is usually too unrealistic for the process we are modelling. In the context of Bayesian optimization, a more realistic alternative is



(a) $l = 0.3$



(b) $l = 3$

Figure 2.4: Gaussian process posterior after six points with different length-scale. On the top with a low length-scale, data points are almost irrelevant, the Gaussian process returns to its prior almost immediately. On the bottom, the GP has a very high confidence in its prediction.

the Matérn 5/2 kernel:

$$k(r) = \sigma^2 \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right) \tag{2.10}$$

The chosen value of $\nu = 5/2$ means that the samples will be twice differentiable, which is a good compromise between too smooth and too difficult to optimize θ, as many point estimate methods require twice differentiability (Snoek, Larochelle, and Adams (2012)). Figure 2.5 shows what samples from these kernels look like.

The presented kernels have the common property of being stationary, i.e they depend only of $\mathrm{x} - \mathrm{x}'$. They are invariant to translation. It is particularly relevant in the context of hyper-parameter optimization as the kernels make no difference

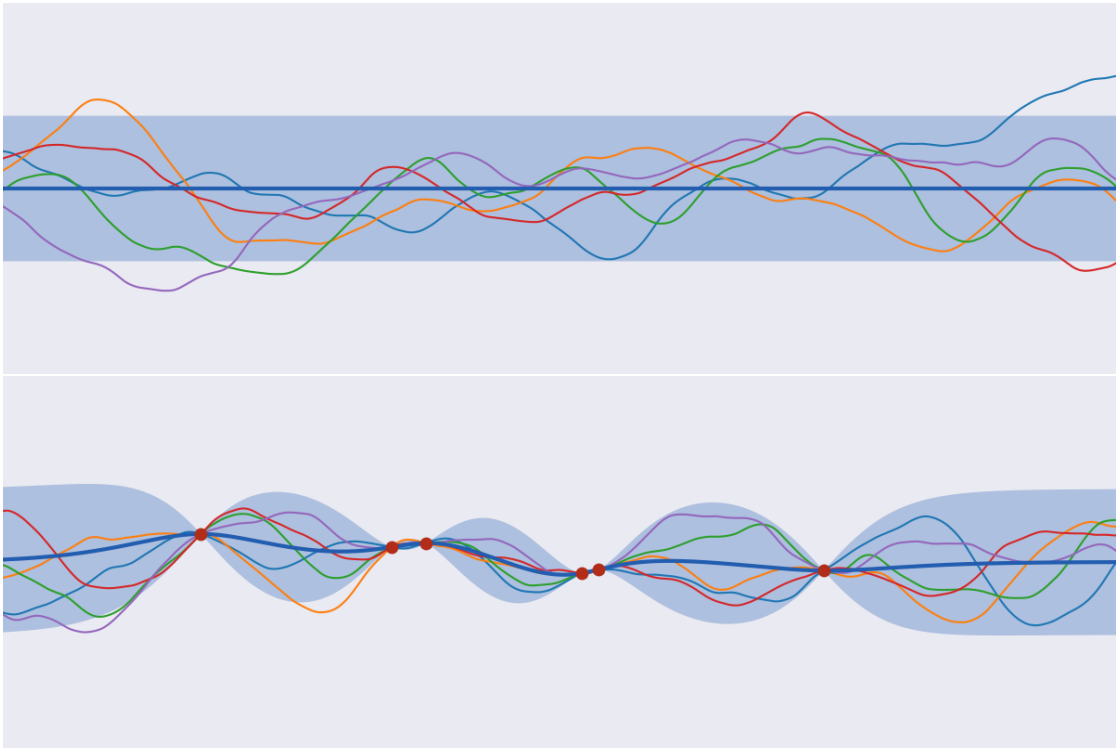


Figure 2.5: Gaussian process using a Matérn 5/2 kernel. On the top, the prior. On the bottom, the posterior after fitting six points.

between values of 3 and 2 or 1000 and 999. For hyper-parameters where such a difference matters should use a non-stationary kernel instead of the ones presented above (see Paciorek and Schervish (2003)).

2.2.1.4 Learning the kernel hyper-parameters

Kernels have themselves hyper-parameters θ which need to be chosen. For example, the squared exponential kernel has l, the characteristic length-scale. As shown by Neal (1996), the inverse of the length-scale determines how relevant an input is. In the context of hyper-parameter optimization, it can help to choose which hyper-parameters to tune carefully. It is therefore very important to select a good value for l.

There are two ways to learn θ. The first way is to maximize the marginal likelihood, which can be derived as:

$$\log p\left(\mathrm{y}|\mathrm{X},\theta\right) = -\frac{1}{2}\mathrm{y}^T K^{-1}\mathrm{y} - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{2.11}$$

The optimization can be done with any off-the-shelf method, eventually with multiple restarts as there are no guarantee of a unique optimum.

The other solution is to not learn θ at all and instead marginalize the hyper-parameters, i.e. at the inference step compute:

$$p\left(\mathrm{y}_*|\mathrm{X}_*, \mathrm{X}, \mathrm{y}\right) = \int p\left(\mathrm{y}_*|\mathrm{X}_*, \mathrm{X}, \mathrm{y}, \theta\right) p\left(\theta|\mathrm{X}, \mathrm{y}\right) d\theta \tag{2.12}$$

This integral is usually intractable but can be approximated by sampling methods. Murray and Adams (2010) use slice sampling, Garbuno-Inigo, DiazDelaO, and Zuev (2016) use asymptotically independent Markov sampling and Titsias, Rattray, and Lawrence (2011) review different Monte Carlo methods used for this problem.

2.2.2 Acquisition functions

In the context of Bayesian optimization, the Gaussian process gives for each set of hyper-parameters an estimation of the performance of the corresponding model and the uncertainty of the Gaussian process in its estimation. But we do not have a way to decide which model is the most interesting to train. Do we pick a model that will be slightly better than our best current model, i.e. where the Gaussian process gives low uncertainty, or do we pick a model with high uncertainty but which could have low performance? There is an exploration/exploitation trade-off to be found. It is the role of the acquisition function to determine which model to train.

The oldest acquisition function is the Probability of Improvement (Kushner (1964)). It chooses the model which has the highest probability of having better results than a target, which is usually picked to be the loss of the current best model. The PI is defined as below, where Φ is the Normal cumulative distribution function, x represents a given set of hyper-parameters, y_* is the minimum loss found so far, μ is the mean returned by the Gaussian process and σ the variance:

$$PI(\mathrm{x}) = \Phi\left(\frac{y_* - \mu(\mathrm{x})}{\sigma(\mathrm{x})}\right) \tag{2.13}$$

The problem with this function is that it is highly sensitive to the choice of target. The simplest choice is the minimum loss found so far, but the PI will then sample models very close to the corresponding model completely ignoring exploration. A better target is how much to improve the minimum loss. But this choice is very

inconvenient because we usually have no idea of how much better the performance can get. Should we try to find a model 1% better? 5%? 25%? If we pick too big an improvement, the function will simply select the models with the highest uncertainty.

Instead, we can use the Expected Improvement function (Schonlau, Welch, and Jones (1998), Jones (2001)) which builds on the Probability of Improvement as below where ϕ is the normal density function:

$$EI(\mathrm{x}) = \sigma(\mathrm{x})[u\Phi(u) + \phi(u)] \tag{2.14}$$

with

$$u = \frac{y_* - \mu(\mathrm{x})}{\sigma(\mathrm{x})} \tag{2.15}$$

The Expected Improvement EI is obtained by taking the expectation of the improvement function (Shahriari et al. (2016)), defined as:

$$I(\mathrm{x}) = (y_* - \mu(\mathrm{x}))\, \mathbb{1}\,(y_* > \mu(\mathrm{x})) \tag{2.16}$$

This function is equal to 0 if the predicted mean is less than the best loss found so far (which means that there is no improvement), otherwise it is proportional to the gap between the predicted mean and best loss.

The Expected Improvement has a major advantage over the Probability of Improvement. The target is always the minimum loss found so far, meaning that there is no need to guess a threshold of improvement.

2.2.3 Bayesian optimization algorithm

In summary, Bayesian Optimization works as follows. Given a set of explored combinations and their associated models performance, a Gaussian process fitted on that set is used to predict the performance of untested combinations. Then, an acquisition function takes those predictions and decides which combination should be tested next. This combination is tested, added to the set of explored combinations, and the process is repeated as long as resources are available. This process can be seen in Figure 2.6 on a one-dimensional problem.

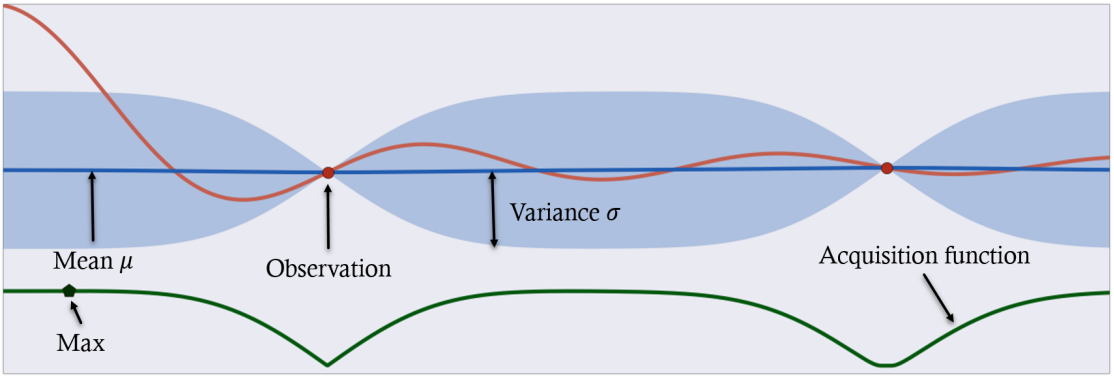


Figure 2.6: Bayesian Optimization on a one dimensional function. The orange curve is the true loss function, the blue one is the prediction of the Gaussian process. The green curve is the acquisition function. The next evaluation will be at the maximum of the green curve.

2.3 Incremental Cholesky decomposition

2.3.1 Motivation

The costliest part of Bayesian optimization is the construction of the Gram matrix K and the computation of its invert K^{-1}. To reduce that cost, the standard solution is to compute the Cholesky decomposition L of the Gram matrix and invert the decomposition. Every time the Gram matrix is modified, the Cholesky decomposition and its inverse are fully recomputed.

However, Bayesian optimization alters the Gram matrix only by adding new rows and columns corresponding to the covariance between the n older combinations and the k new combinations. The Gram matrix can therefore be decomposed in blocks where one of them is the previous Gram matrix. By calling $K_{(n,n)}$ the previous matrix and $K_{(n+k,n+k)}$ the new one with the added points, the decomposition is:

$$K_{(n+k,n+k)} = \begin{pmatrix} K_{(n,n)} & K_{(k,n)}^T \\ K_{(k,n)} & K_{(k,k)} \end{pmatrix} \tag{2.17}$$

Likewise, the Cholesky decomposition and its inverse can also be decomposed into blocks where one is the previous Cholesky decomposition as we show in Section 2.3.2. We then argue in Section 2.3.3 that this can be used to make Bayesian optimization faster by updating incrementally the Cholesky decomposition instead of fully recomputing it each time.

2.3.2 The incremental formulas

The derivation of both formulas can be found in Appendix A. We present here only the final results. The formula of the incremental Cholesky decomposition is:

$$L_{(n+k)} = \begin{pmatrix} L_{(n)} & 0 \\ K_{(k,n)}(L_{(n)}^T)^{-1} & L_{(k)} \end{pmatrix} \tag{2.18}$$

Since this formula requires the costly computation of $L_{(n)}^{-1}$, we are also interested in an incremental formula for it, which is:

$$L_{(n+k)}^{-1} = \begin{pmatrix} L_{(n)} & 0 \\ K_{(k,n)}(L_{(n)}^T)^{-1} & L_{(k)} \end{pmatrix}^{-1} = \begin{pmatrix} L_{(n)}^{-1} & 0 \\ -L_{(k)}^{-1}K_{(k,n)}(L_{(n)}^T)^{-1}L_{(n)}^{-1} & L_{(k)}^{-1} \end{pmatrix} \tag{2.19}$$

2.3.3 Complexity improvement

A standard Cholesky decomposition has a complexity of $O\left((n+k)^3\right)$ with $n+k$ the number of observed combinations. With our formulas, since we already have $L_{(n)}$, all that is left is to compute $L_{(k)}$, which has a cost of $O\left(k^3\right)$. Since Bayesian optimization is called after every tested combination, $k = 1$, the Gram matrix just has one new row and one new column. $L_{(k)}$ and its inverse are trivial to compute.

There is however an increased cost in memory, as $L_{(n)}$ and $L_{(n)}^{-1}$ must be stored between calls of Bayesian optimization. They are two $n \times n$ triangular matrices, storing them cost $O\left(n^2\right)$, which seems a reasonable price to pay for the performance improvement.

2.4 Comparing Random Search and Bayesian Optimization

In this section we study the theoretical performance of random search, before devising an experiment that compares the performance of random search and Bayesian optimization in a practical setting.

2.4.1 Random search efficiency

How many models of a hyper-parameters space should be trained in order to be reasonably certain to have trained one of the best models? If we knew the loss l_{min}

of the best model, how long would it take to find a model with a loss such that $l \leq (1+\alpha)l_{min}$? Due to the relative simplicity of random search, we can derive theoretical bounds to answer these questions.

Let N be the total number of models in the hyper-parameters space, M the number of models satisfying our performance criteria (alternatively, the M best models of the space) and n the number of models to train.

Considering that random search chooses models uniformly, the probability of drawing one of the best models the first time is simply $\frac{M}{N}$. The second time, it is $\frac{M}{N-1}$ since we do not allow redrawing a model. We can therefore define the probability of not drawing any satisfying models after n draws as the following equation, where Y is the random variable of failing to draw an acceptable models among n draws in a row.

$$P\left(Y=n\right) = \prod_{k=0}^{n}\left(1-\frac{M}{N-k}\right) \tag{2.20}$$

This is a particular case of the hypergeometric distribution. From there, the probability of drawing an acceptable model at the n-th draw is:

$$P\left(X=n\right) = \frac{M}{N-n}\prod_{k=0}^{n-1}\left(1-\frac{M}{N-k}\right) \tag{2.21}$$

X is the random variable of drawing an acceptable model after $n-1$ bad draws. The probability of drawing an acceptable model in the first n draws is:

$$P\left(X\leq n\right) = \sum_{k=0}^{n} P\left(X=k\right) \tag{2.22}$$

From this equation we can compute the number of draws required to have drawn a model in the top $\alpha\%$, by setting M as a fraction of N. Moreover since the equation depends only of the ratio $\frac{M}{N-k}$, the effect of k becomes negligible as N grows bigger and the equation converges.

In Table 2.2, we present some of those results, which allow us to draw the following strong conclusion. For any hyper-parameter space, random search will draw a model in the top 5% of models in 90 draws with a probability of 99%. Less than a hundred draws is sufficient to draw one of the best models of any hyper-parameter space. It is a surprisingly low number given the simplicity of the method.

	$p > 0.5$	$p > 0.95$	$p > 0.99$
Top 1%	69	298	458
Top 5%	14	59	90
Top 10%	7	29	44

Table 2.2: Number of draws required to have drawn a model in the top $\alpha\%$ with probability p in a space of 100 000 combinations.

However that is only one of the questions we asked. Ranking the models by their performance does not guarantee that a model in the top $\alpha\%$ is within $\alpha\%$ of the performance of the best model (i.e. whose performance verifies $l \leq (1+\alpha)l_{min}$), unless model performance happens to be uniformly distributed. Since we have no idea of how model performance is distributed, we cannot go further in our theoretical analysis. In Section 2.4.3, we return to this question by studying empirically the distribution of model performance.

2.4.2 Bayesian optimization efficiency

While we were not able to obtain similar results for Bayesian optimization as we did for random search, we expose some of the difficulties in doing so.

First any analysis depends of the exact setup of Bayesian optimization. A change of kernel or acquisition function changes the results. But the main problem is that, because the process learns the structure of the hyper-parameter space, any analysis would need to work in a particular space.

None of those difficulties prevent us from measuring the empirical performance of Bayesian optimization, as we show in the next section.

2.4.3 Experiments on CIFAR-10

2.4.3.1 Setup

In order to compare the performance of the different hyper-parameter optimization methods, we devised a hyper-parameter space containing 600 models and trained them all on the CIFAR-10 dataset (Krizhevsky (2009)).

The dataset is composed of 60 000 images equally divided in 10 classes. The images are 32x32 RGB images. The training set contains 50 000 images, the test set 10 000.

Since the goal of the experiment is not to beat the state-of-the-art on the dataset, the selected hyper-parameter space and the resulting models are small

and simple. Each model is trained for a total of 10 minutes on a NVIDIA TITAN X card.

We then compare in which order the different methods selected the models. The methods are evaluated on the time needed to select the best model, as well as the time needed to select a model in the top $\alpha\%$.

However one run is not enough to conclude that Bayesian optimization on average is more efficient than random search. Due to the huge computing cost, we cannot do hundreds of runs with different seeds and retrain each network every time. But if we do not retrain the network but change the seeds of the search policy, we can simulate hundreds of runs at a low cost.

There are a total of 600! way to explore this space, but because we do not retrain the models, all the randomness in Bayesian optimization is in the choice of the first model[2], leaving us with only 600 possible paths through this space. We computed them all, then randomly selected 600 ordering for random search.

2.4.3.2 Models distribution

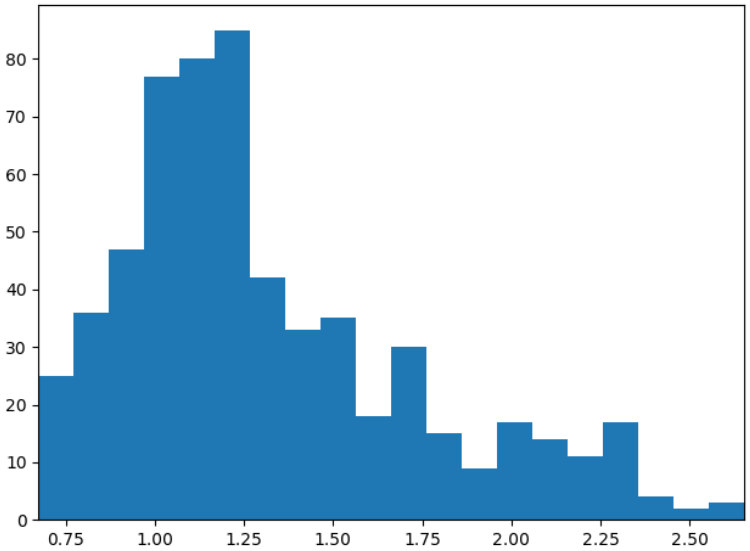


Figure 2.7: Distribution of the models performance.

Before comparing the search methods, we look at the distribution of models performance in Figure 2.7. A few models are really good, most are average and

[2]This is true for our specific approach. Incorporating Gaussian noise in the observations ($\hat{y} = y + \mathcal{N}(\mu, \sigma)$)) for example would make our analysis incorrect.

then there is a long trail of progressively worse models. It looks like a skew normal distribution.

We cannot extrapolate that every hyper-parameter space will have a distribution like this, but we assume for now that this is the case for hyper-parameter spaces designed in practice.

Since the distribution is not uniform, there is a difference between models in the top 5% of models and models that have a performance within 5% of the performance of the best model. In this case, there are 30 models in the top 5% of models (since there are 600 models), but only 6 are within 5% of the performance of the best model. In this case, the 30-th best model is within 18% of the best model.

This distinction is important because it changes our evaluation of the methods. We are not simply interested in the average time to find a model in the top α% of models, but also in the average time to find a model within α% of the best model.

2.4.3.3 Results

First, we look at the order in which models where selected during one run, shown in Figure 2.8. The blue points represent the test loss of the models in the order the methods selected them, and the red line shows the minimum loss found at a given time. There is a clear trend present for Bayesian optimization where the combinations evaluated later have low performance, suggesting it did learn to predict correctly the performance of untested combination. In this run, it was also able to find the best model much earlier than random search. Random search behaves as expected and models performance is uniformly distributed.

	Random Search		Bayesian optimization	
	Average	Worst	Average	Worst
Best model	297 ± 171	599	87 ± 64	249
Within 1%	146 ± 114	554	26 ± 16	120
Within 5%	87 ± 75	397	17 ± 11	67
Within 10%	54 ± 52	296	15 ± 9	44
Top 1%	87 ± 75	397	17 ± 11	67
Top 5%	18 ± 17	106	10 ± 8	38
Top 10%	10 ± 10	62	7 ± 7	36

Table 2.3: Average and worst number of draws taken by each method to reach different goals over 600 runs. Within α% mean within α% of the performance of the best model.

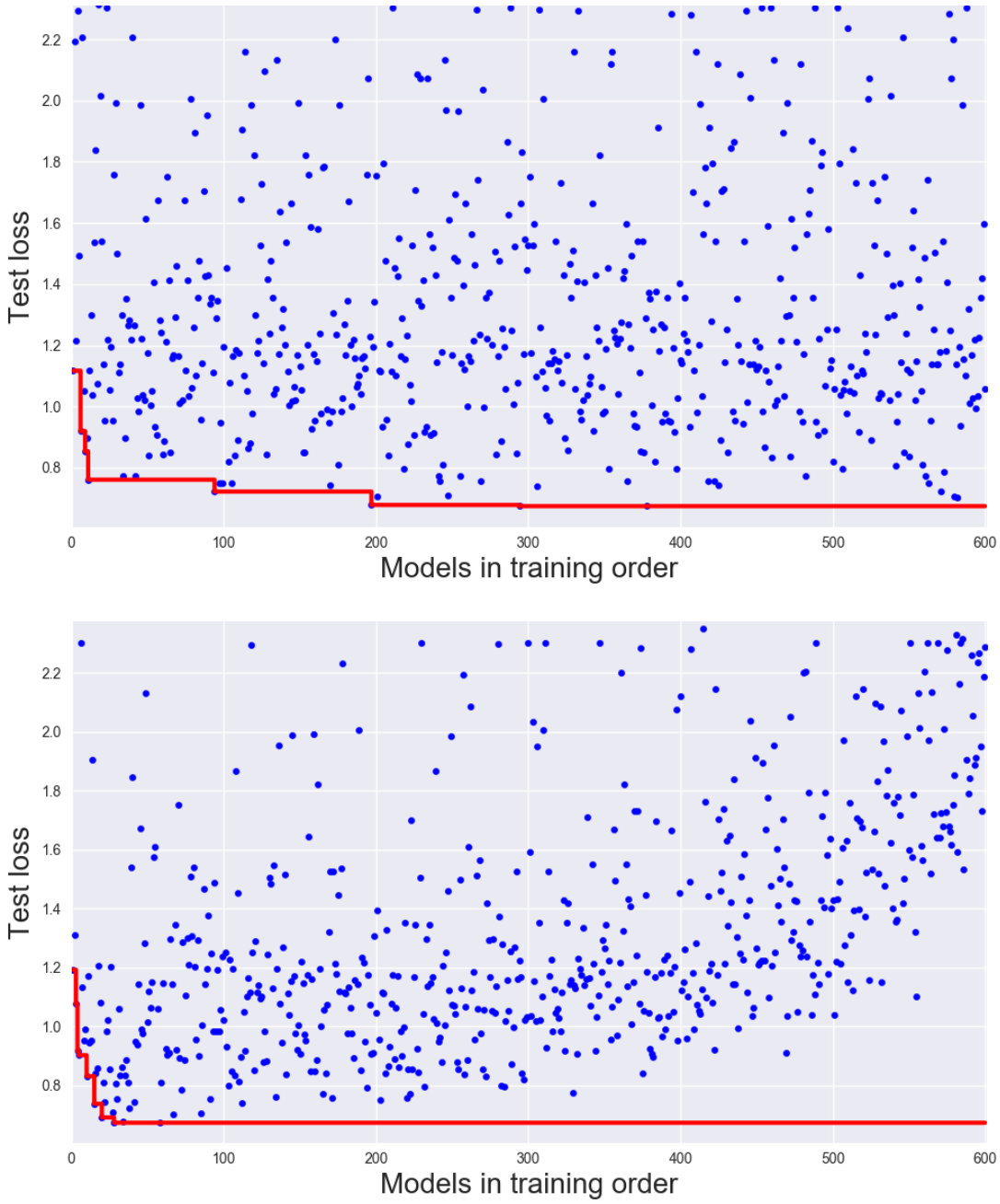


Figure 2.8: Comparing the model order chosen by random search (top) vs Bayesian optimization (bottom).

To confirm these trends, we look at the average number of draws taken by each method to reach different goals (Table 2.3).

Random search needed on average 297 draws before finding the best model, which is within the expected range of the theoretical value of 300 draws. If we rank the model by their performance, random search needed an average of 18 draws to find a model in the top 5% and 87 to find a model in the top 1%.

In comparison, Bayesian optimization did a lot better on every goal. On average it needed 87 draws to find the best model and only 17 to find a model in the top 1%. In most cases the worst run of Bayesian optimization performed better than the average run of random search.

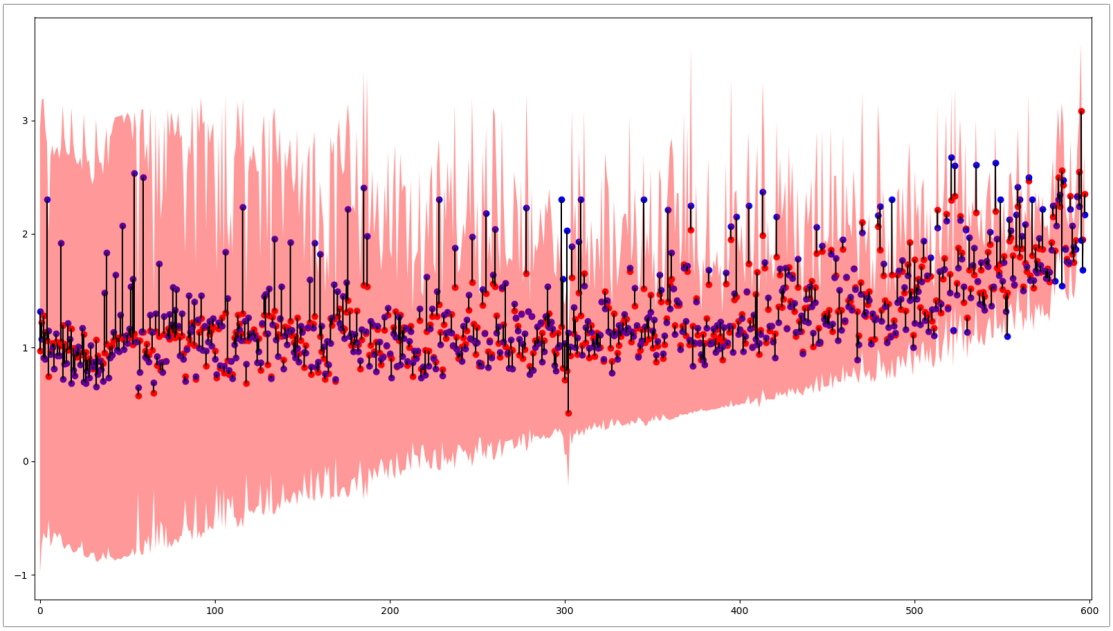


Figure 2.9: Model performance predicted by the Gaussian process in red vs the true performance in blue. The colored area is the 95% prediction interval.

To confirm whether the Gaussian process is learning the structure of the hyper-parameter space, Figure 2.9 shows the predicted performance and the true performance, as well as the 95% prediction interval. As more models are used to refine the Gaussian process, the prediction interval shrinks, i.e. the Gaussian process becomes more confident in its prediction. Prediction for outliers, in this case models that do not learn, is highly inaccurate at the beginning and is still the main source of error at the end.

2.4.4 Conclusion

We designed an experiment to measure and compare the performance of random search and Bayesian optimization on a setting mimicking real life usage. Performance of random search matched closely theoretical results, showing the surprising efficiency of this method. Bayesian optimization outperformed random search on every metric, which justifies its much higher implementation cost.

Extending the hyper-parameter space would yield insights into how Bayesian optimization behaves in larger spaces. This framework of testing could be used to observe the behaviour of other methods such as Hyperband. We observed models performance to be normally distributed, but the scope is limited to one hyper-parameter space on one task. To the best of our knowledge there is no theory on how to build good hyper-parameter spaces and understanding the relation between models, tasks and model performance would be of practical use to the design of hyper-parameter optimization methods.

2.5 Combining Bayesian Optimization and Hyperband

This section describes and extends the work presented at CAp 2017 (Bertrand, Ardon, et al. (2017)). We propose a new method of hyper-parameter optimization that combines Bayesian optimization and Hyperband.

2.5.1 Hyperband

A property of neural networks is that their training is usually iterative, usually some variant of gradient descent. A consequence is that it is possible to interrupt training at any time, evaluate the network, then resume training. This is the property Hyperband (Li et al. (2017)) takes advantage of.

The principle is simple: pick randomly a group of configurations from a uniform distribution, train the corresponding networks partially, evaluate them, resume training of the most performing ones, and continue on until a handful of them have been trained to completion. Then pick a new group and repeat the cycle until exhaustion of the available resources.

But a problem appears: at which point in the training can we start evaluating the models? Too soon and they will not have started to converge, making their evaluation meaningless, too late and we have wasted precious resources training

under-performing models. Moreover, this minimum time before evaluation is hard to establish and changes from one task to the other. Some hyper-parameters (such as the learning rate) even influence the training speed! Hyperband's answer is to divide a cycle into brackets. Each bracket has the same quantity of resource at its disposal. The difference between brackets is the point at which they start evaluating the models. The first bracket will start evaluating and discarding models very early, allowing it to test a bigger number of configurations, while the last bracket will test only a small number of configurations but will train them until the maximum number of resources allowed per model.

The algorithm is controlled by two hyper-parameters: the maximal quantity of resources R that can be allocated to a given model, and the proportion of configurations η kept at each evaluation. R is typically specified in number of epochs or in minutes. At each evaluation, $1/\eta$ models are kept while the rest are discarded.

2.5.2 Combining the methods

Bayesian optimization and Hyperband being orthogonal approaches, it seems natural to combine them. Hyperband chooses the configurations to train uniformly and intervenes only during training. On the other side, Bayesian optimization picks configurations carefully by modelling the loss function, then let them train without interruption.

As a result, Hyperband does not improve the quality of its selection with time, while Bayesian optimization regularly loose time training bad models.

Combining the methods fixes these problems. Model selection is done by Bayesian optimization as described in Section 2.2, then Hyperband train them as described in Section 2.5.1. Two changes are required for Bayesian optimization, as it needs some way to distinguish between fully-trained models and partially trained models. To do that, the training time becomes an hyper-parameter, making the performance of each model at every minute a distinct point. The performance prediction is then done at the training time corresponding to Hyperband's first evaluation, to have only one prediction per model.

The second change is to normalize the values returned by the acquisition function to make it a probability distribution, i.e. divide each value by the sum of all values, so that the sum of all normalized values is equal to one. In the standard usage of Bayesian optimization, the chosen model is the argmax of the acquisition function. Choosing the top α models yields models that are very close as the

acquisition function is smooth and gives similar values for nearby combinations. This strategy removes the property of Hyperband to explore many regions of space at the same time. Normalizing the values and drawing from this distribution keeps this property, while making the selection smarter over time as it changes to reflect the knowledge acquired from the trained models.

The proposed algorithm is as follows: the first group of configurations is chosen randomly and evaluated according to Hyperband. All subsequent selections are done by training a Gaussian process with a squared-exponential kernel on all evaluated models. The expected improvement is then computed on all untested combinations and normalized to make a probability distribution from which the next group of models is sampled.

2.5.3 Experiments and results

We compare the three methods presented above: Bayesian optimization, Hyperband, and Hyperband using Bayesian optimization. The algorithms were implemented in Python using scikit-learn (Pedregosa et al. (2011)) for the Gaussian processes and Keras (Chollet et al. (2015)) as the deep learning framework.

Comparison was done on the CIFAR-10 dataset (Krizhevsky (2009)), which is a classification task on 32×32 images. The image set was split into 50% for the training set used to train the neural networks, 30% for the validation set used for the Bayesian model selection and the rest as test set used for the reported results below.

Each method is allocated a total budget $B = 4000$ minutes meaning the sum of all the models training time must be equal to 4000 at the end of the process. The choice of having a budget in time means that models will not be trained in epochs as usual, but in minutes. This is a practical choice that allows estimating accurately the total time that the search takes, though it has the effect of favoring small networks. Indeed, two models trained an equal amount of time will not have seen an equal amount of data if one model is bigger and thus slower than the other. The choice to constrain in time instead of epoch means that the quantity of data seen by the models depends on the GPU. The training is done on two NVIDIA TITAN X.

The chosen architecture is a standard convolutional neural network with varying number of layers, number of filters and filter size per layer. Other hyper-parameters involved in the training method are: the learning rate, the batch size and the presence of data augmentation. In total there are 6 hyper-parameters to

Hyper-parameter	Range of values
Number of convolutional blocks	$[1; 5]$
Number of convolutional layers per block	$[1; 5]$
Number of filters per layer	$[2; 7]$
Filter size	$\{3; 5; 7; 9\}$
Learning rate	$\{10^{-5}; 10^{-4}; 10^{-3}; 10^{-2}\}$
Batch size	$[2; 9]$

Table 2.4: Hyper-parameter space explored by the three methods.

tune for a total of 19 200 possible configurations, displayed in Table 2.4.

For Hyperband, we chose $R = 27$, meaning 27 models are chosen at each iteration and are trained for a maximum of 27 minutes, and $\eta = 3$ which means that 1/3 of the models are kept at each evaluation. In the case of Bayesian optimization, each model was trained 30 minutes but they are chosen sequentially.

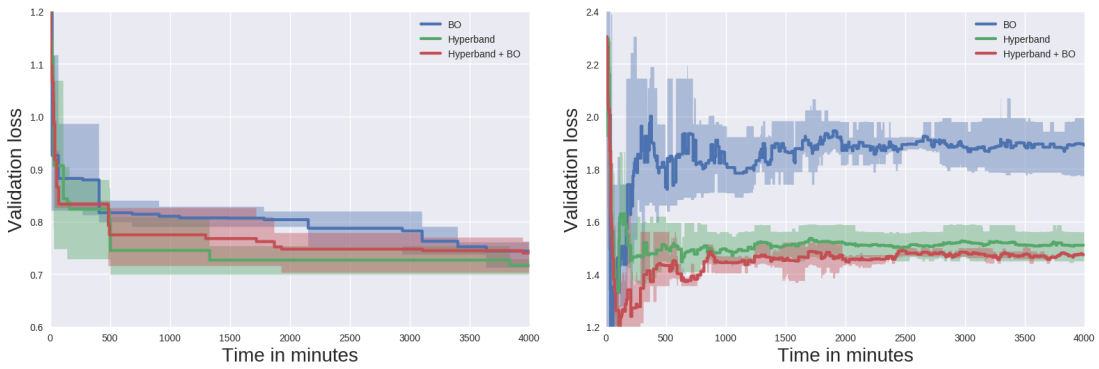


Figure 2.10: (Left) Loss of the best model found at a given time by each method. (Right) Running median of the loss of all tested models for each method.

The evaluation measure that matters when comparing methods is the loss of the best model found at a given time, and is illustrated in Figure 2.10 for the two individual methods and their combination (5 runs each). The running median suggest that Bayesian optimization (in blue) performs slightly worse than both other methods, and that Hyperband with Bayesian optimization (in red) finds a better model quicker than Hyperband (in green). However, due to the stochastic nature of the methods, many more runs would be needed to draw definite conclusions from this evaluation measure.

It is more informative to look at the running median of the loss of all models trained at a given time, as it gives us an idea of the quality of the models tried as

the methods progress. Ideally the running median should go down for Bayesian optimization and Hyperband as more models are tested and the methods either learn the shape of the space or stop training inefficient models. Bayesian optimization performs notably worse than the other two methods. Hyperband and Hyperband + BO have similar performance, though Hyperband + BO seems slightly better.

2.5.4 Discussion

Due to the part of chance in all three methods, five runs are not enough to draw definite conclusions on their performance. Ideally hundreds of runs would have been needed, and the computational cost of this endeavor was the main reason we did not pursue further this problem. At 4000 minutes per method per seed, the above experiment already required around 42 days of GPU time without accounting for the overhead (loading the data, building the models, ...).

While the general idea of combining Hyperband and Bayesian optimization is valid, there are two problems with our specific approach.

As mentioned previously, since Bayesian optimization is usually sequential, we normalized the results of the acquisition function to make it a probability distribution from which the models to train were sampled. However the resulting distribution was very close from a uniform distribution. This is because the expected improvement outputs values in a small range, i.e. the difference between the maximum and the minimum is at most a few orders of magnitude. When normalized, those extreme values would not be particularly likely, even with just a small hyper-parameters space of 19200 combinations. For example if the highest value is a hundred times the average value, after normalization, the corresponding model has only around 0.5% chance of being sampled. This is why our version of Hyperband + BO performed barely better than Hyperband alone.

One potential solution is to use the "liar" strategy. After choosing a model but before training it, it is added to the training set with a low performance and the acquisition function is re-computed. The "lie" of the low performance discourages the acquisition function from giving high values to the surrounding models, and the argmax will be a completely different model. Repeating this process allows drawing many models simultaneously.

When drawing a large group of models at the same time as required by Hyperband, the cost of this process becomes important as the Gram matrix of the Gaussian process needs to be inverted many times. The incremental Cholesky decomposition presented in Section 2.3 could be used to make the process faster.

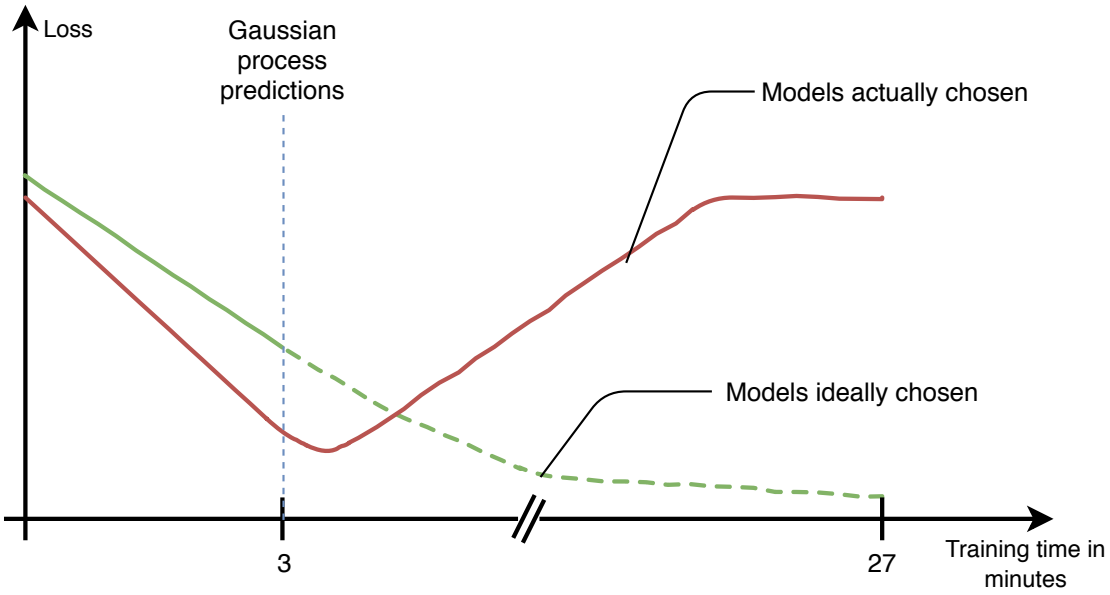


Figure 2.11: The problem with predicting model performance at 3 minutes (dotted blue line). Models that trained faster (due to being smaller, red line) have better performance at 3 minutes than bigger models (green line), however they overfit shortly after while the bigger models end up with a better performance.

Informal observations of the chosen models revealed a second problem. After a few iterations, many of the models chosen tended to have converged to their best performance by the time of Hyperband's first evaluation (after 3 minutes of training) and to start overfitting immediately after (red line in Figure 2.11). This was due to how Bayesian optimization handled the training time. Since all models were trained at least 3 minutes, but only some trained longer, the Gaussian process predictions were much more accurate at 3 minutes than at 27 minutes. This was our reason to use the prediction at 3 minutes, despite this side-effect.

While using the prediction at 27 minutes instead would avoid choosing the smaller models that overfit, it would also discard a lot of information as the predictions of all the models trained only partially would have reverted to the prior of the covariance function by 27 minutes. It would be better to design a special kernel to deal with only this hyper-parameter and that does not revert quickly. Inspiration for this kernel could be found in Domhan, Springenberg, and Hutter (2015) as these authors study models to extrapolate learning curves.

Using a different strategy, this combination method would perform better than either Hyperband or Bayesian optimization, as shown recently in Falkner, Klein, and Hutter (2018).

2.6 Application: Classification of MRI Field-of-View

This section describes and extends work presented at ISBI 2017 (Bertrand, Perrot, et al. (2017)). It aims at applying the hyper-parameter optimization studied earlier in this chapter to a real application: the classification of MRI field-of-view.

2.6.1 Dataset and problem description

Given an MRI volume we would like to know which anatomical regions are contained in it and their position in the volume. Since MRI data are acquired in a slice by slice approach, we reduce the problem to a two dimensional classification task of axial slices (into head, chest, abdomen, pelvis, legs and spine).

The dataset consists of MRI images coming from a variety of hospitals and machines across the world (such as the *Centre Hospitalier Lyon-Sud, France* or *Johns Hopkins University, USA*). As a consequence the images are highly varied in terms of protocols (see Figure 2.12) as well as resolution, number of volumes

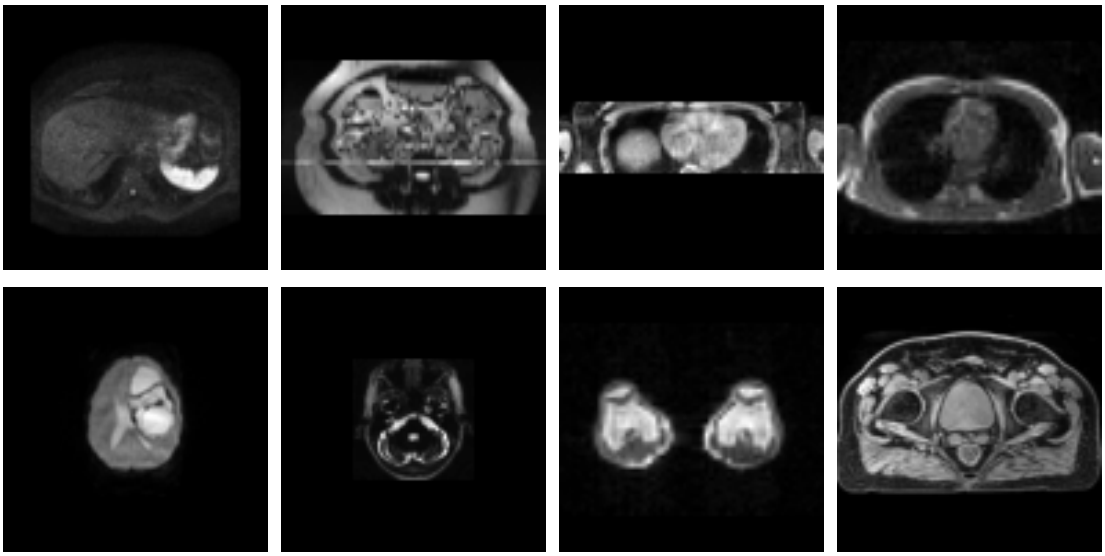


Figure 2.12: Images from the MR dataset showing the variety of classes and protocols.

Body Part	# Volumes	# Slices
Head	404	17011
Chest	186	6897
Abdomen	367	15486
Pelvis	351	17962
Legs	99	12868
Spine	390	9790

Table 2.5: Content of the MRI field-of-view dataset.

per class and number of slices per volume. Table 2.5 sums up the content of our dataset.

The dataset is splitted into a training set for the optimization of the weights of the networks, a validation set for model selection (optimization of the hyper-parameters) and a test set for model evaluation (respectively 50%, 25%, 25% of the database). The separation is done volume-wise to take into account intra-subject slices correlations. Volumes containing multiple classes are split by anatomical regions and can end up in different sets. This raises the difficulty of the task since, in case of overfitting, predictions will be wrong at validation or testing phases.

Finally, each slice is subject to a unique step of preprocessing: it is resized to 128×128 pixels, a good trade-off between time constraints and quality of information.

Data augmentation is used and consists in generating 80 000 images per epoch. The augmentation is done by applying translations, shearing and rotations, zooming, and adding Gaussian noise.

2.6.2 Baseline results

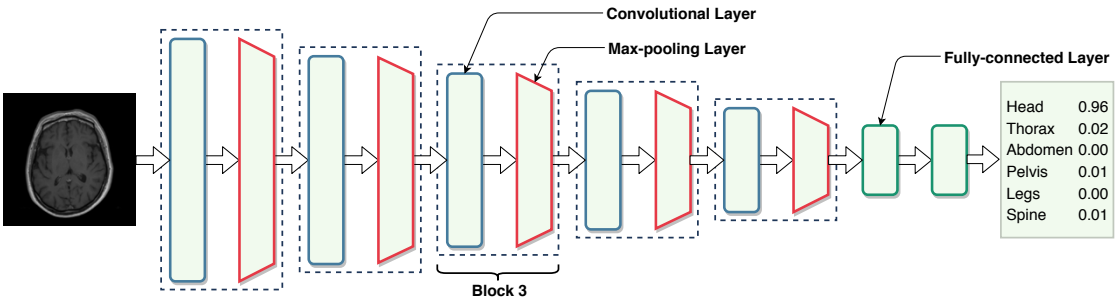


Figure 2.13: Baseline network used to solve the problem.

Starting from a standard VGG architecture (Simonyan and Zisserman (2014)), we modified it manually until settling on the following model, shown in Figure 2.13: it is divided into 5 blocks, each comprising a convolution layer of 64 filters of size 3×3, followed by a rectified linear unit (ReLu) and a max-pooling layer. The network ends with 2 fully-connected layers (respectively 4096 and 1024 units) interleaved with ReLU activations and terminated with a softmax decision layer. This network was trained by minimizing the categorical cross-entropy loss weighted by class frequency, using stochastic gradient descent (SGD) with a learning rate of 10^{-3}, Nesterov momentum ($m = 0.9$) and decay ($d = 10^{-6}$) for 30 epochs.

We achieve a 14% error rate with most of the error focused on the chest. Examining the confusion matrix (Table 2.6), we note that the abdomen, pelvis and legs images have most of their errors located with anatomically adjacent regions. Further examinations of the misclassified images revealed they were for the most part located at the boundaries of their classes. These errors make sense as these boundaries are ill-defined and varies from one volume to the next.

2.6.3 Hyper-parameter optimization

By relaxing some structural parts of our baseline architecture we define a large family of models. This hyper-parametric family has the following structure (see Figure 2.14): (1) b convolution blocks, each including c convolutional layers of 2^r

		Predicted					
		Head	Chest	Abdomen	Pelvis	Legs	Spine
Actual	Head	96	0	1	2	1	0
	Chest	1	57	13	28	1	0
	Abdomen	0	1	88	10	0	1
	Pelvis	1	0	9	81	8	1
	Legs	0	0	0	19	81	0
	Spine	0	1	3	2	0	94

Table 2.6: Confusion matrix for the baseline on the test set, in percent.

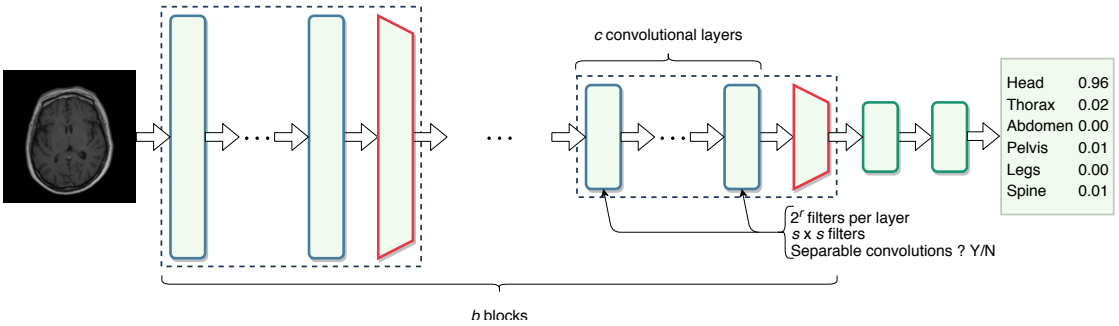


Figure 2.14: Variations on the baseline allowed in our search.

filters of size $s \times s$ interleaved with ReLU activations and terminated by a max-pooling layer, (2) the fully-connected layers as in our baseline architecture, and (3) a final softmax decision layer.

The last hyper-parameter is whether the convolutions are separable or not. Separable convolutions are two different convolutional layers using 1×3 filters then 3×1. This leads to a 33% reduction in the number of weights often without negative impact on the model performance.

Changes within this parametric space of models may drastically transform the optimization landscape, requiring to adjust the training setting accordingly (in our case: learning rate and batch size, all other settings remaining identical). These architecture hyper-parameters and training settings form the hyper-parameter space. The ranges of those hyper-parameters, detailed in Table 2.7, were defined so as to fulfill memory (less than 12GB) and time constraints (training should last less than one day).

Name	Range	Baseline
Number of blocks	$b \in [1; 5]$	5
Number of convolutional layers per block	$c \in [1; 5]$	1
Number of filters per convolutional layer	$2^r, r \in [2, 8]$	64
Size of the filters	$s \in \{3; 5\}$	3
Separable Convolutions	$e \in \{\text{Yes}, \text{No}\}$	No
Learning Rate	$10^l, l \in [-7; 0]$	0.001
Batch Size	$2^a, a \in [2; 8]$	8

Table 2.7: Description of the hyper-parameters, with their range and and the baseline value.

While many more hyper-parameters could have been considered, the ones we chose already define a search space of 34 300 models, well-above our capacity to fully explore. Those hyper-parameters were chosen as they were considered to have the highest impact on the performance by a mix of intuition and experience. A more systematic approach would likely yield better results, though at a greater computational cost.

To explore this hyper-parameter space, we used Bayesian optimization, as described in Section 2.2.[3] We chose the Expected Improvement as our acquisition function, and we used the loss on the validation set as the value the Gaussian process must predict. Every network was trained for 30 epochs, and we trained a total of 300 models. The first model picked by the Bayesian optimization is usually chosen randomly, here it made sense to start from the baseline as it was already trained.

At the end, our optimization process yields a collection of models ranked by their performance. The quality of this assessment is naturally limited since the size of the validation set used in this respect cannot encompass the diversity of clinical reality. Cross-validation could be used to get a better estimator of the performance, but we cannot practically afford its costs.

Figure 2.15 shows the performance of the models in the order they were trained. The baseline is quickly improved upon and the best model is found after 33 iterations. Another model with similar performance is found after 55 iterations. Starting at iteration 90, most of the chosen models are too big to fit in memory and therefore unable to train. While the fact that no previously chosen models were too memory intensive but suddenly most chosen models are may seem

[3]While the previous section suggests Hyperband is a better method, this work was done before Hyperband.

Figure 2.15: Performance of the models in the order they were trained. Each dot is a model. The red line represents the performance of the best model found until this point. The models with a loss of 2.5 are models that could not be trained as they were too big.

surprising, this can be explained for two reasons.

The first model of the process was our hand-crafted baseline, which had good performance. This likely allowed for a good initialization of the Gaussian process, and the following models explored the area around the baseline. As models explored further and further than the baseline, eventually one was chosen that was too big to fit in memory. This argument is speculation that should be confirmed by repeating the Bayesian optimization many times with a different initialization every time. Lack of resources prevented us from exploring this point further.

The second argument explains why most models became impossible to train as soon as the first of those was chosen. This is due to how we handled models impossible to train for memory limitations. We chose to simply remove such models from the search space. However the acquisition function still gives a high priority to this region of space and a surrounding model is chosen that is also likely to be too big. Since we remove those models on subsequent iterations, the acquisition function never learns to not go there.

An alternative way to handle un-trainable models would have been to choose

		Predicted					
		Head	Chest	Abdomen	Pelvis	Legs	Spine
Actual	Head	94	0	1	1	2	0
	Chest	0	93	5	1	1	0
	Abdomen	0	2	85	10	2	1
	Pelvis	1	2	4	86	6	1
	Legs	0	4	1	19	76	0
	Spine	0	1	1	0	2	96

Table 2.8: Confusion matrix for the best model on the test set, in percent.

an arbitrarily high loss for those models. This violates the smoothness assumption of the Gaussian process and would have affected strongly the prediction on nearby models, discouraging the acquisition function from choosing them. This would have also disqualified nearby models that fit in memory, which was our reason for not doing that in the first place.

It is not clear which strategy is better. Choosing a high loss sacrifices some valid models, but saves time by testing less un-trainable models. On the other hand, an un-trainable model fails immediately at model creation, so the gain in time is minor. But independently of the chosen strategy, the fact that some combinations lead to model too big to fit in memory highlights a flaw in the design of the hyper-parameters space and the resulting networks architectures. The un-trainable models are models with a low number of blocks but a high number of filters per layer. With one block and 64 filters per layer, the feature maps of the last convolutional layer are of size $64 \times 48 \times 48$, meaning the following fully-connected layer have $64 \times 48 \times 48 \times 4096 = 603,979,776$ weights. Since each block ends with a max-pooling layer, the number of weights quickly becomes manageable with additional blocks.

The best model found with Bayesian optimization has an error rate of 10%, an improvement of 4% over the baseline. Looking at the confusion matrix in Table 2.8, there is a significant improvement in the classification of chest, going from 57% accuracy in the baseline to 93% accuracy. The other classes have similar accuracy as the baseline. Most of the errors are abdomen and legs slices located at the boundaries of the pelvis.

Looking at the architectures of the two best models that have very similar error

rates, it can be observed that the first one only deviates from the baseline by having two convolutional layers per block but only 32 filters per convolutional layer and a higher batch size of 32. The second model however is completely different. It has only 4 blocks with 3 convolutional layers per block, uses separable convolutions, with a smaller batch size of 4 and learning rate of 10^{-4}.

The last important factor to consider is the total number of weights. The baseline model had $6,713,226$ weights, the best model had $5,468,778$, a 19% reduction. The second best model had $13,927,114$ weights due to using one less block than the other two models. Once again most of the weights are caused by the transition from the last convolutional layer to the first fully-connected layer. To improve these results the first step would be to better design this transition, either by controlling the number of max-pooling layers or by replacing the fully-connected layers with convolutional layers, as is typical of modern convolutional neural networks.

2.6.4 From probabilities to a decision

Even though we made the choice of processing the data at the slice level, the slices form volumes and we are interested in a decision at the volume level. The first question is: does the model classifies two successive slices similarly? This is expected since successive slices are anatomically close and have a similar structure. If the model gives very different predictions, this could be a sign that it relies on noise patterns instead of the anatomical structures.

To answer this question, we processed a full body volume by classifying each of its slices through our best model. As we can see in Figure 2.16, the network is able to identify all body parts, despite the slices being processed independently. Nevertheless the network tends to misclassify the boundaries between regions, notably legs/pelvis and pelvis/abdomen. It also mistakenly identifies the empty slices above the head as being pelvis with a high confidence. Interestingly, the straps used to hold the patient down, visible as the bands with high intensity on the extremities and low intensities on the body, trigger the network every time into the pelvis class. This is likely due to the rarity of those straps in the dataset.

But we are not interested in the mere probabilities, we want to take an actual decision. Therefore we need a decision scheme. A very simple one would be a threshold, say 0.7, over which we consider the slice to be part of the class. However the predictions from the network are too noisy (see Figure 2.16-b), and this approach gives regions broken in multiple parts with messy boundaries.

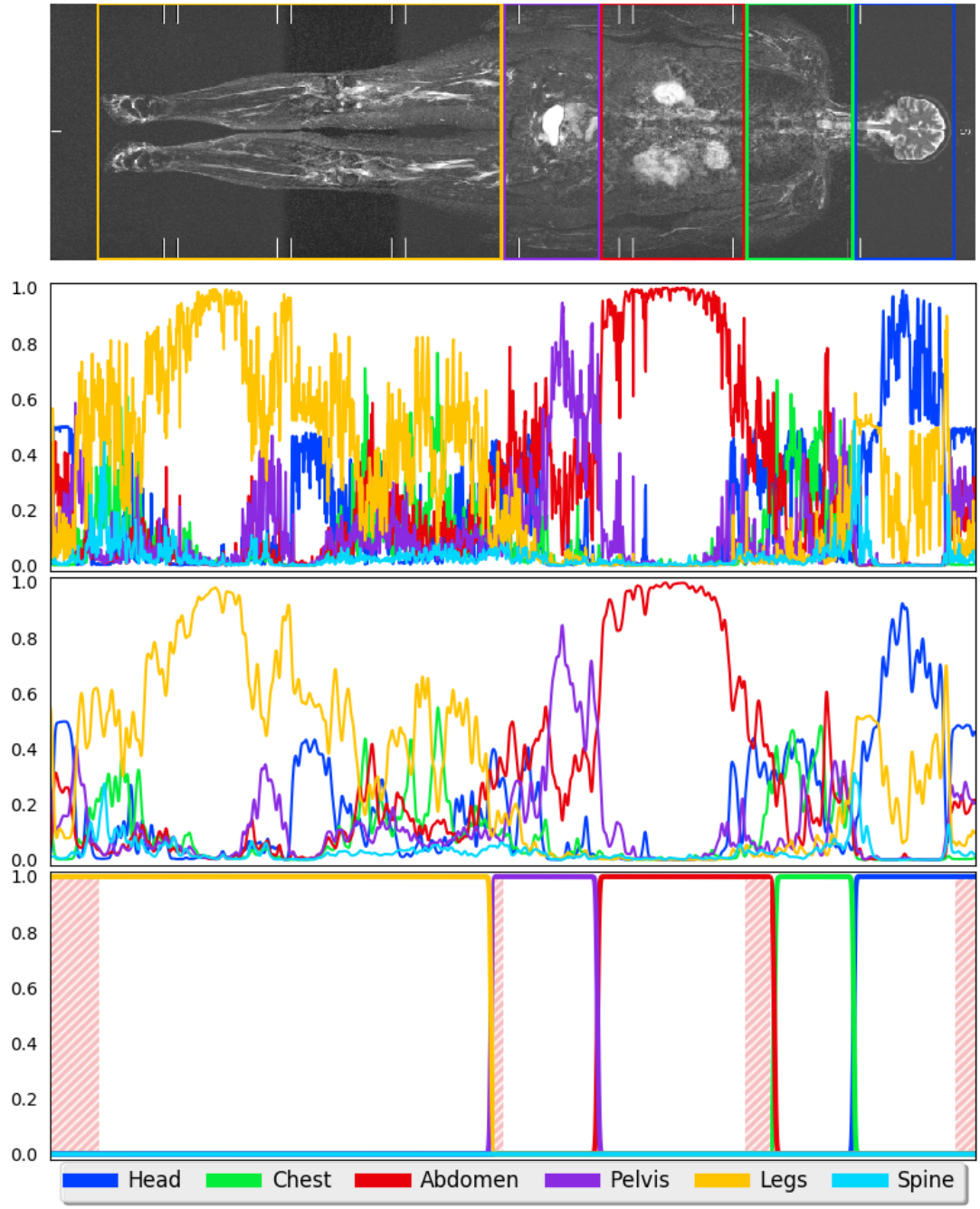


Figure 2.16: Slice by slice prediction on a full body volume. From top to bottom: (a) Image and ground truth. (b) Raw model prediction. (c) After smoothing. (d) Final prediction, after selecting the best transitions. Highlighted in red are the errors compared to the ground truth.

A first step is to smooth the probabilities curves with a Gaussian filter, as illustrated in Figure 2.16-c. This makes the predictions easier to read, and we can see on this example that the predictions are correct almost everywhere except for the chest. Additionally, the pelvis is preceded and followed by the abdomen, which is of course impossible.

From this last observation, we note that our goal is to find contiguous anatomical regions, delimited by a starting slice Z_i^m and an ending slice Z_i^M, where i is the index of the region in anatomical order (legs - pelvis - abdomen - chest - head) and Z is the slice number. Those boundaries must be such that $\forall i \in [1; N], Z_{i+1}^m \geq Z_i^M$ (a region cannot start before the end of the previous region). The network outputs the probability of a slice to belong to a region $P_i(Z)$.

We translate these constraints on region boundaries and the fact that intuitively, the best region between two slices is the one that maximizes a region probability, through the following constrained linear program:

$$\begin{aligned} &\text{minimize} \quad -\sum_{i=1}^{N} \int_{Z_i^m}^{Z_i^M} P_i(z)dz \\ &\text{subject to} \quad Z_{i+1}^m \geq Z_i^M, \qquad i = 1, ..., N \end{aligned} \tag{2.23}$$

The Lagrangian is:

$$\mathcal{L}\left(Z_i^m, Z_i^M, \lambda\right) = -\sum_{i=1}^{N} \int_{Z_i^m}^{Z_i^M} P_i(z)dz - \sum_{i=1}^{N} \lambda_i \left(Z_{i+1}^m - Z_i^M\right) \tag{2.24}$$

From the partial derivatives, we obtain:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial Z_i^m} &= P_i\left(Z_i^m\right) - \lambda_{i-1} \\ \frac{\partial \mathcal{L}}{\partial Z_i^M} &= -P_i\left(Z_i^M\right) + \lambda_i \end{aligned}$$

Setting the derivatives at zero, the optimal boundary slices must verify:

$$\lambda_i = P_i\left(Z_i^M\right) = P_{i+1}\left(Z_{i+1}^m\right) \tag{2.25}$$

This result implies that the optimal boundary slices are the slices where adjacent regions intersect. For example, a slice where the probability of belonging to the pelvis is equal to the probability of belonging to the abdomen is a potential boundary slice, but an intersection slice between pelvis and head is not. We show an

example of valid and invalid transitions slices in Figure 2.17.

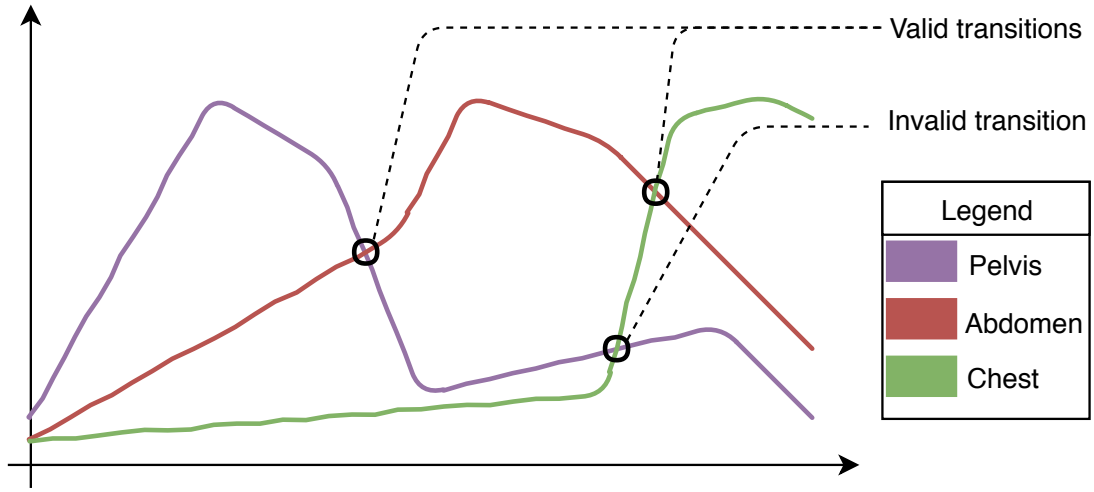


Figure 2.17: Example of valid and invalid transitions. Pelvis/abdomen and abdomen/chest are valid as they are from adjacent classes, but chest/pelvis is invalid.

To find the optimal boundary slices, we find all the valid intersection slices, construct all the valid sets of transitions and compute the function defined in Equation 2.23. The minimum is the optimal set of boundary slices. Following the example in Figure 2.16-d, the final prediction is wrong only at the extremities of the volume (our decision scheme forces to pick a class) and is in minor disagreement with the ground truth legs/pelvis boundary. The only error of consequence is the abdomen/chest boundary. Compared to the raw prediction which was wrong for the pelvis and chest, the improvement is clearly visible.

This scheme is robust because of its properties: each region is at most one contiguous block (it can also be not present), the regions are always in anatomical order and it allows for many misclassified slices.

2.6.5 Conclusion

We have shown how to use Bayesian optimization to the concrete problem of MR FOV classification. The gain in performance between our handcrafted model and the best model clearly shows the value of automating the search of hyperparameters, even with a limited budget.

The search was vulnerable to all the weaknesses of Bayesian optimization: the process was sequential (only one model trained at a time), we were limited in the

hyper-parameters we could use (no conditional hyper-parameters) and the kernel was stationary.

Another problem encountered is how to deal with models that cannot be trained (because they require too much memory for example). Not putting them in the training set will just let the search pick other similar models, but putting an arbitrarily high value is not ideal either as the Gaussian process will smoothly interpolate to that value, even though there is a discontinuity in that region (between the models that can and cannot be trained).

Regarding the final models, testing at the volume level has shown that the models are robust and can generalize. Still, the neural network was not enough to obtain smooth coherent predictions, and we presented a decision scheme to fix that.

Further work on this problem would involve improving the dataset by adding new volumes, new anatomical regions and defining stricter boundaries between regions. It could also be interesting to explore how to directly predict the boundaries of the regions using deep learning, instead of classifying each slice and using another method to obtain the regions.

3 Transfer Learning

Abstract

After a brief tour of the landscape of transfer learning methods in Section 3.1, we introduce the problem of kidney segmentation in 3D ultrasound images across two populations: healthy adults and sick children (Section 3.2), which will be the main application of the methods proposed. Using standard transfer learning methods as our baseline in Section 3.3 was insufficient for our problem and led us to develop a new transfer learning method presented in Section 3.4. That method is based on predicting transformations to be applied to images during inference. We wrap up the chapter with a comparison of the tested methods and a discussion of the results in Section 3.5 and a conclusion in Section 3.6.

Contents

3.1 The many different faces of transfer learning

Transfer learning is the idea of re-using knowledge learned in one situation for another situation. This makes intuitive sense in many frequently encountered cases. To take examples from the medical field, a model trained on the detection of a specific disease in a specific imaging protocol should have learned something about these images that could be used for the detection of another disease. Those images have common properties like the intensity of organs and bones that computer vision models learn and use. Transfer learning aim to avoid re-learning those properties but instead transfer them from one model to the other. Models may also learn other kinds of knowledge, such as the anatomy of the human body. Organs do not change shape or location from one imaging protocol to the other, and models may learn and use this information.

But to be useful, transfer learning must be faster or more convenient than simply training a model from scratch. In the medical field in particular, the acquisition of data is costly and time-consuming, and labeling or segmenting the acquired data to build a training database even more so. It is very common to have only a handful of patients from a particular population. That scarcity of data may make some tasks unsolvable by themselves, but pooling together every available bit of data may allow solving them.

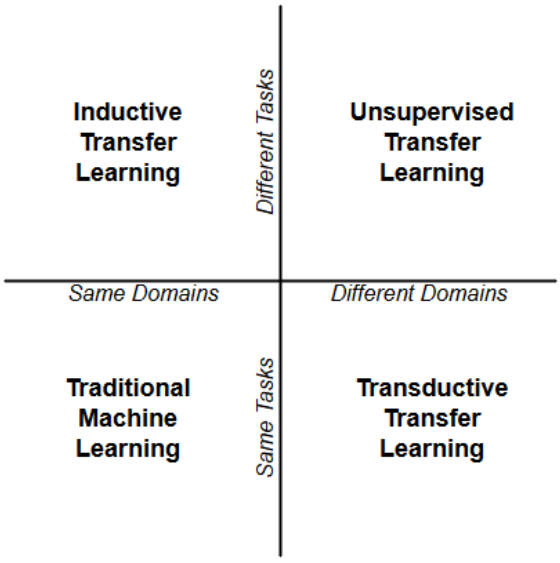


Figure 3.1: Transfer learning can be divided into four categories depending on how the domains and the tasks are related.

Following the notation of Pan and Q. Yang (2010), we define the *domain* as a feature space $\mathcal{X}$ and a probability distribution $P(\mathrm{X})$ such that $\mathrm{X} = \{x_1, \cdots, x_n\} \in \mathcal{X}$. A *task* is composed of a label space $\mathcal{Y}$ and an objective predictive function f which is learned from the training data (i.e. pairs of $\{x_i, y_i\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$). The transfer is done from a *source* domain and task to a *target* domain and task. These notions form the two axes according to which transfer learning methods are divided. How are the source and target domains related, and how are the source and target tasks related? These axes result in four different problems as shown in Figure 3.1.

Inductive transfer learning is the category of problems where the source and target domains are identical, while the source and target tasks are different, but related. An example of such a problem would be an image database where the source task is to detect the presence or absence of particular objects and the target task is to locate those objects in the image. Transductive transfer learning is the category where the source and target domains are different but related, while the source and target tasks are identical. An example that we introduce in Section 3.2 is the segmentation of the kidney in 3D ultrasound images, where the source domain is composed of healthy adults and the target domain is composed of sick children. Unsupervised transfer learning happens when both the domains and the tasks are different. In this setting, labels are unavailable for both source and target. In all three cases, transfer of knowledge is possible by exploiting the nature of the differences between the domains and the tasks.

In this section we briefly present these three kinds of transfer learning. Then Section 3.1.4 focuses on transfer learning in the context of deep learning.

If both the domains and the tasks are identical, this is simply standard machine learning and not transfer learning.

3.1.1 Inductive Transfer Learning

In this setting, the source and target tasks are different, while the source and target domains stay the same. Many approaches have been explored in this setting. An extension of the AdaBoost algorithm, TrAdaBoost (Dai et al. (2007)), assumes that data in the source and target domains share features and labels but have different distributions, in order to propose an iterative reweighting of the source domain data during training.

Learning a low-dimensional representation of the data that is shared across tasks is an idea explored in a supervised fashion (Argyriou, Evgeniou, and Pontil

(2006), Lee et al. (2007)) and in an unsupervised fashion (Raina et al. (2007)).

It is also possible to transfer parameters or hyper-parameters of various models. Lawrence and Platt (2004) proposed to learn kernel parameters for Gaussian processes across tasks, Evgeniou and Pontil (2004) developed a regularization framework for sharing parameters of SVMs. Gao et al. (2008) built an ensemble learning framework where the weights of each model are assigned dynamically for each sample in the target domain.

3.1.2 Transductive Transfer Learning

Transductive transfer learning (Arnold, Nallapati, and Cohen (2007)), also called domain adaptation, focuses on solving the same task across different domains. The difficulty comes when little data is available in the target domain, or when labels are unavailable or sparse for the target domain. A lot of work has been done on how to relate the unlabeled target data with the labeled data through the loss of the model (Arnold, Nallapati, and Cohen (2007))) or in which proportion they should be mixed during training (Crammer, Kearns, and Wortman (2008), Ben-David et al. (2010)).

When the model learns simultaneously on the different domains, this is multi-domain learning (Daume III (2007), M. Joshi et al. (2013)).

3.1.3 Unsupervised Transfer Learning

There has been relatively little work in this setting. Dai et al. (2008) proposed a *self-taught clustering* algorithm which aims to cluster a small unlabeled target dataset at the same time as a big unlabeled source dataset. By learning a common feature space, clustering on the target dataset is improved.

Chang et al. (2018) developed a method to learn a bank of multi-scale convolutional filters from a large unlabeled source dataset and a small target dataset, which can then be used in a convolutional neural network to solve classification tasks. Only the construction of the filters bank is done without labels. Applications on various biomedical tasks show a clear improvement in performance over using only the target dataset.

3.1.4 Transfer Learning and Deep Learning

One method has become the *de facto* face of transfer learning for deep neural networks, due to its simplicity and versatility: fine-tuning (Hinton, Osindero, and

Teh (2006), Bengio et al. (2007)). The idea is that the features learned by a neural network are generic enough, so that it becomes interesting to re-use them on a different domain, for a different task, or both. No modifications to the architecture are done. The only difference is that the weights of the network are not initialized randomly, but set to the final weights learned on the source domain.

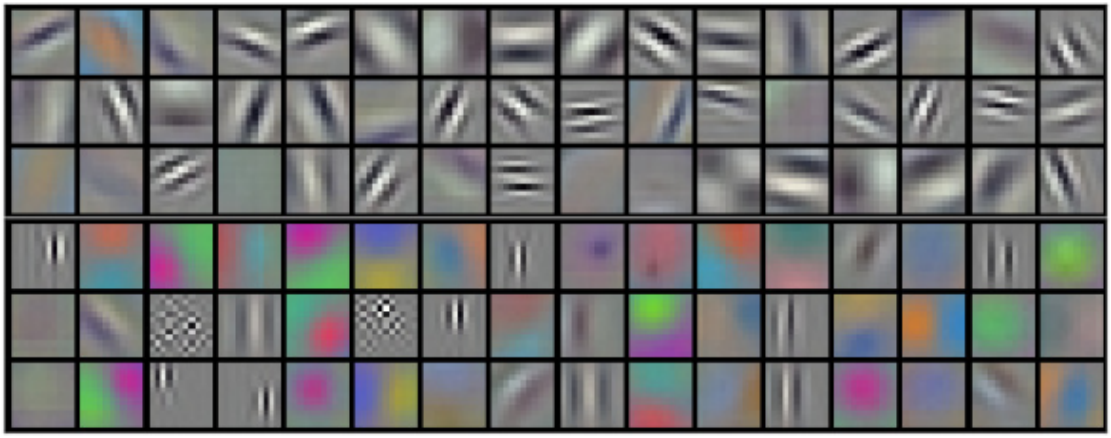


Figure 3.2: The first convolutional layer of AlexNet learned many filters very similar to Gabor filters. (Source: Krizhevsky, Sutskever, and Hinton (2012))

It is not always necessary, or even a good idea, to re-train the whole network. If the target dataset is too small, full fine-tuning risks overfitting. But then, which layers should be fine-tuned? Lower layers tend to be more generic, while higher layers tend to be specialized to the task (Yosinski et al. (2014)), suggesting that it would be better to fine-tune only the last n layers of the network. As illustrated in Figure 3.2, the AlexNet architecture trained on ImageNet learns filters very similar to Gabor filters in its first convolutional layer, showing their genericity. An additional advantage is that it is faster as there is no need to back-propagate to the lower layers. The exact value of n is entirely problem-specific, the closer the domains and the tasks are, the less layers need to be fine-tuned.

Moreover the source dataset does not need to be closely related to the target dataset. A network trained for classification on the ImageNet dataset (Russakovsky et al. (2015)) can be fine-tuned on other natural images dataset for classification or even detection and improve the results over training from scratch (Oquab et al. (2014), Razavian et al. (2014)).

An alternative to fine-tuning is feature extraction (Donahue et al. (2014), Sermanet et al. (2014)). It is similar to fine-tuning in that it uses the fact that the features learned by the lower layers of a neural network are fairly generic. But

instead of modifying the higher layers of the network, here they are discarded and the output of the lower layers are used as features for a completely different model, not necessarily a neural network (see Figure 3.3). Features extracted from natural images have been shown to be useful to medical images (Shin et al. (2016), Bar et al. (2015), Ginneken et al. (2015)).

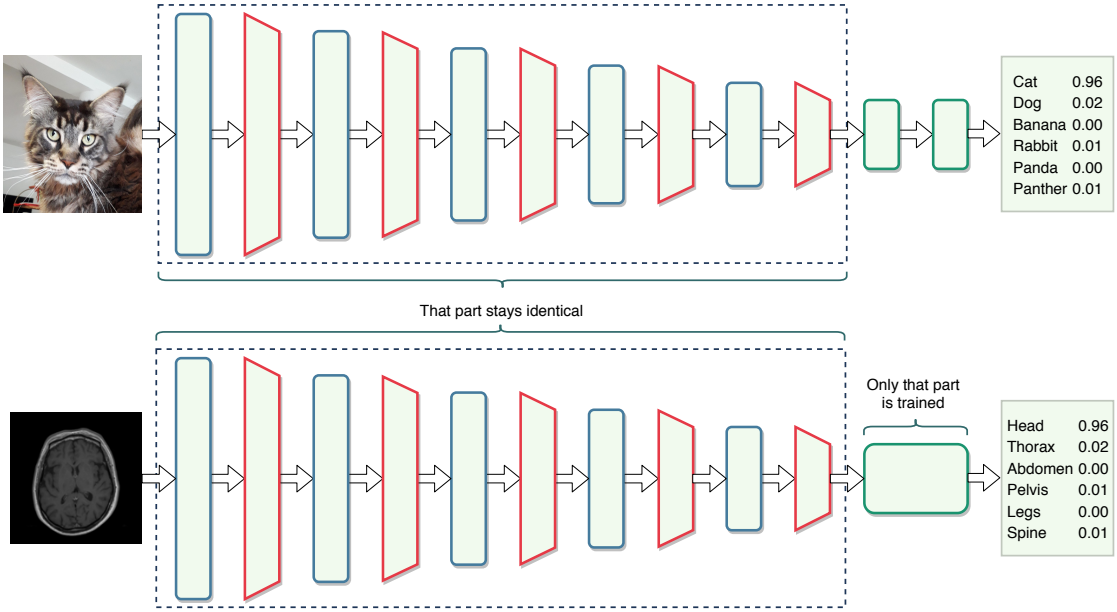


Figure 3.3: Feature Extraction. The network at the top is trained on natural images. It can be reused on medical images by extracting the features at a given layer of the network and using them as the input of a new model.

Autoencoders (Ballard (1987), Bengio et al. (2007)) are neural networks learned in an unsupervised manner. The goal of the network is to reconstruct its input from a compressed representation of the data that it will learn. The first part of the network progressively compresses the input while the second part decompresses it. This first part can then be used as a pre-trained network for other datasets and tasks (Bengio et al. (2007), Vincent et al. (2008), Erhan et al. (2010)). The representation learned by the autoencoder can also be used as the features of another model, which is helpful even for distant domains such as natural images and MR images (Gupta, Ayhan, and Maida (2013)). More generally, unsupervised pre-training on natural images has successfully been used to improve results on medical imaging tasks (Schlegl, Ofner, and Georg Langs (2014), Hofmanninger and G. Langs (2015)).

The methods discussed so far assume a sequence of actions. First a source task

is solved on a source domain, and then a target task is solved on a target domain. But if the different tasks are defined from the start, it makes sense to solve them simultaneously. This is the idea of multitask training (Caruana (1995), Caruana (1997), Collobert and Weston (2008)). It consistently gives better results on both tasks than sequential transfer methods.

Likewise, when solving a task on different domains accessible from the start, it makes sense to use the different domains at the same time. The simplest method is to train the network on the union of the different domains, but this does not work well when one domain is a lot bigger than the other. Weighting each sample according to the number of samples of each domain (Daume III (2007)) or balancing batches across domains during training (Buda, Maki, and Mazurowski (2018)) fixes this problem. More complex multi-domain training methods exist, such as Nam and Han (2016) where the network learns domain-specific convolutional layers.

3.2 Kidney Segmentation in 3D Ultrasound

3.2.1 Introduction

The clinical problem motivating this work is the kidney capsule segmentation in 3D ultrasound data from potentially ill children. These images have a high amount of noise, strong dependency on the operator, and important organ shape variability induced by subjects age and degree of illness. Moreover we have limited access to such volumes. In contrast, a much larger database of ultrasound kidney volumes of healthy adults is available, from which we build a successful segmenting deep network. We aim to transfer the knowledge (anatomy, US signal, ...) captured by this network to the pediatric data.

This work has contributions in both the clinical and technical domains: (i) a deep learning based algorithm for kidney capsule segmentation in 3D ultrasound data (Section 3.3), and (ii) a new approach to perform domain adaptation of a pre-trained neural network (Section 3.4). In addition we present experiments comparing different transfer methods according to performance, training time and added complexity to determine the trade-offs of each method (Section 3.5), and discuss their results, examining the segmentation errors to suggest potential improvements.

3.2.2 Related Work

Despite the clinical potential of 3D kidney ultrasounds, both the number of publications and the data used within are relatively small. Moreover, to the best our knowledge, deep learning techniques have not yet been applied. Cerrolaza et al. (2014) and Marsousi, Plataniotis, and Stergiopoulos (2017) have addressed the capsule segmentation problem in 3D pediatric data using active shape models and for real-time segmentation using implicit deformable models, respectively.

Nevertheless, deep learning techniques have been applied by Ravishankar et al. (2017) to 2D ultrasound kidney images. Their approach is compelling since they manage to combine deep learning and shape priors by using two networks, first a U-Net for the segmentation, then a convolutional auto-encoder as a shape prior used for regularization.

The baseline to our work (Section 3.3.1) is the well-known deep learning architecture "U-Net" (Ronneberger, Fischer, and Brox (2015)) and its 3D version (Çiçek et al. (2016)).

3.2.3 Dataset

The ultrasound volumes in our study come from several clinical sites around the world. As a result, there is a high variability in the quality of the images, which translates into various amounts of noise and shadow artifacts as can be seen in Figure 3.4.

We have a total of 503 healthy adults images, and 64 children (aged between 6 months and 15 years), of which about 30% have hydronephrosis in various stages. The database was split into 80% of the images used for training, the rest for testing.

The volumes we give to the network have been down-sampled to a resolution of $4 \times 4 \times 4$ mm^3, and centered in a $80 \times 80 \times 80$ voxels cube. The size in voxels was chosen in order to fit the network and a volume on a single NVIDIA TITAN GPU. The resolution was chosen so that each volume would fit entirely in the cube. We show images before and after pre-processing in Figure 3.4 for the adults and Figure 3.5 for the children.

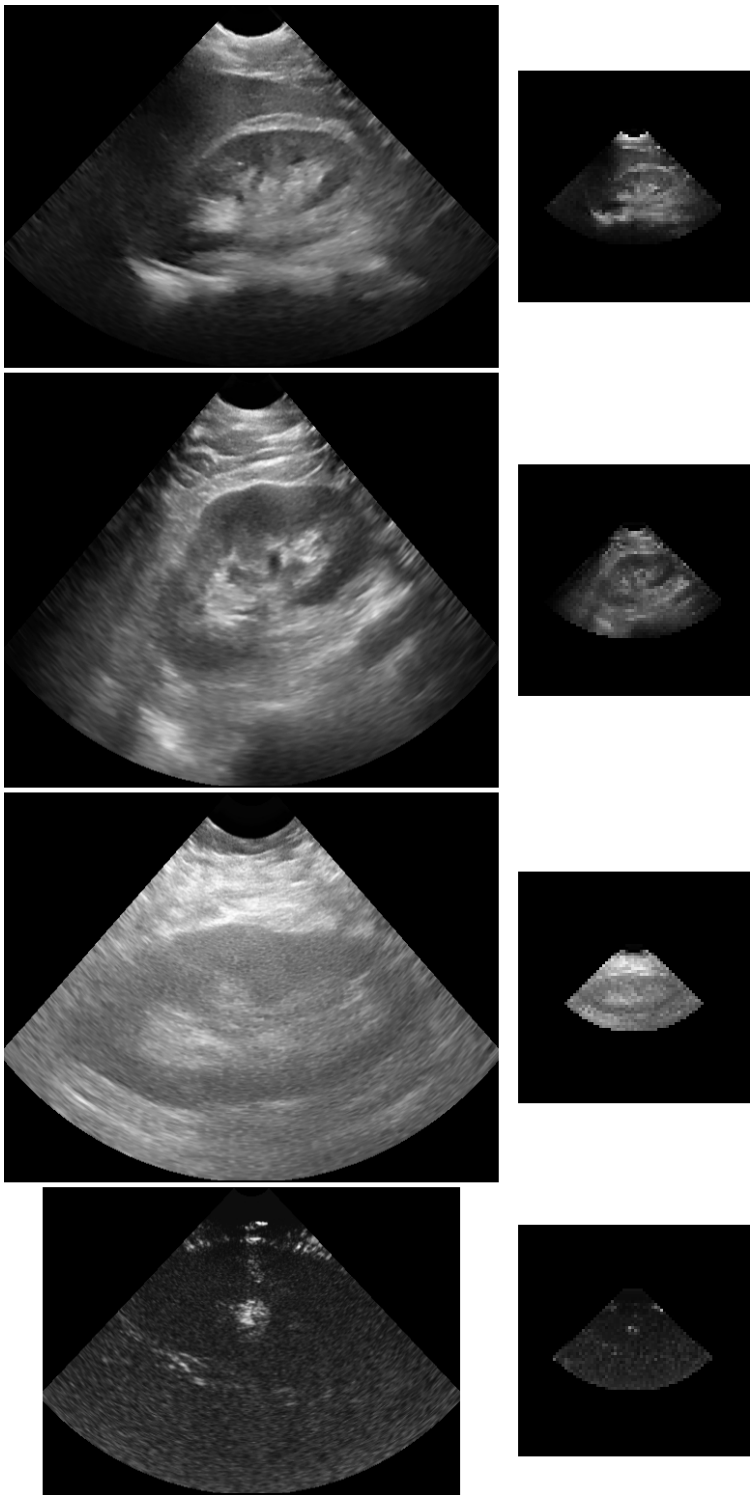


Figure 3.4: Healthy adults kidneys before and after pre-processing.

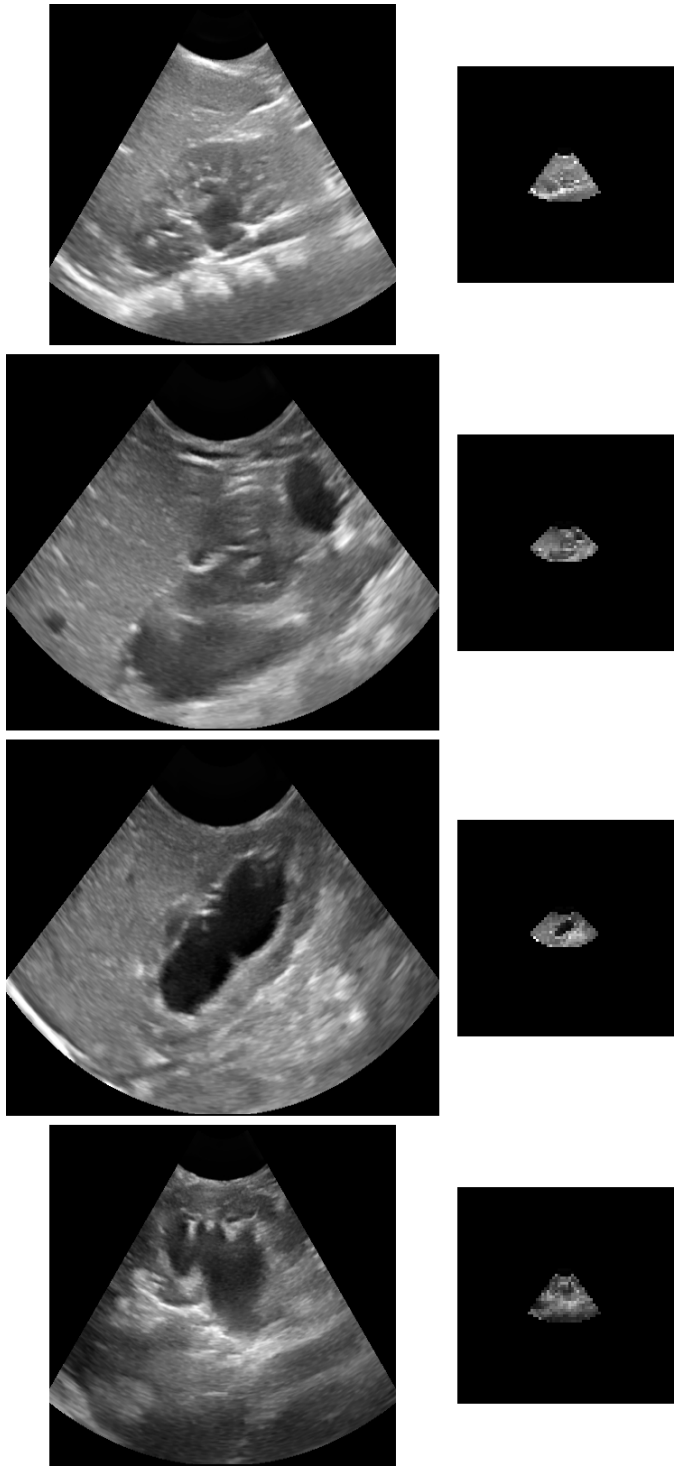


Figure 3.5: Sick children kidneys before and after pre-processing.

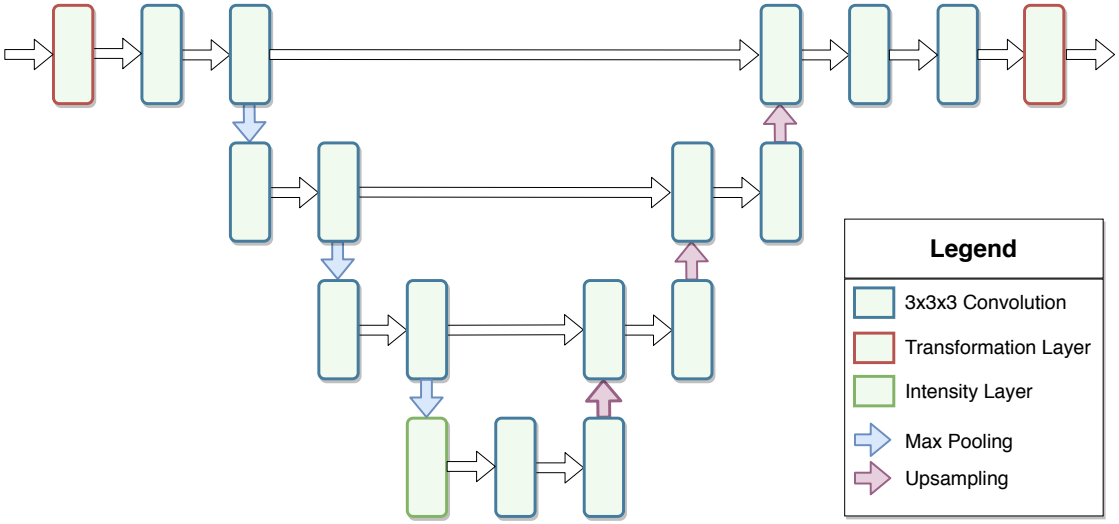


Figure 3.6: U-Net structure with our added transformation and intensity layers.

3.3 Baseline

3.3.1 3D U-Net

The 3D extension (Çiçek et al. (2016)) of the U-Net (Ronneberger, Fischer, and Brox (2015)) architecture is our starting point. Its symmetrical structure alternates convolutional layers with down-sampling layers on one side and convolutional layers with up-sampling layers on the other (see Figure 3.6). At each level, down-sampled feature maps are concatenated to up-sampled ones to give multi-scale information to the network. In our adaptation of this architecture we replace the weighted softmax loss by a Dice-based loss, more adapted to our segmentation task, which is defined as follow:

$$DICE\left(Y, \hat{Y}\right) = \frac{2|Y \cap \hat{Y}|}{|Y| + |\hat{Y}|} \tag{3.1}$$

where Y is the ground truth segmentation and $\hat{Y}$ is the predicted segmentation. The Dice varies between 0 and 1, where 0 means the prediction and the ground truth are disjoint and 1 means a perfect prediction.

Moreover, introducing Spatial Dropout (Tompson et al. (2015)) after each convolution-sampling block proved to significantly improve our results by preventing over-fitting. Whereas standard Dropout (Srivastava et al. (2014)) randomly sets pixels in the feature maps at zero, Spatial Dropout randomly sets filters at

zero, making it more relevant for convolutional layers.

3.3.2 Training the baseline

We used Keras (Chollet et al. (2015)) with the Tensorflow backend (Martín Abadi et al. (2015)). We performed each experiment on five seeds, which impacts both the separation of the data set into training and test sets, and the initialization of the weights of the networks.

Three models are used as baseline: one trained on adults kidneys only, which will also be used for transfer, one trained on children only, and one trained on both, with oversampling of the children to balance each set. Oversampling is a way of dealing with class imbalance (Buda, Maki, and Mazurowski (2018)) and consists in balancing each epoch to have as many adults as children by drawing each children multiple times.

These baseline models allow us to judge the quality of each transfer method. A good transfer method should perform better (or close to) on the children kidneys than the children only network, while staying close to the performance of the adults only network on the adults. The joint model represents the ideal performance we can hope for on adults and children. Results are displayed in Table 3.1, and illustrated in Figure 3.10.

3.3.3 Fine-tuning

The adult baseline network is fine-tuned on the children dataset, resulting in an important drop of performance on the adults and a comparable improvement on the children (see Table 3.1). This is obtained by fine-tuning the whole network.

It is however unnecessary as we show below. Fine-tuning only the last 7 convolutional layers of the network or the first 8 is enough to obtain the same results. This corresponds to all the layers before or after the first upsampling layer. Between the two options, fine-tuning the last layers is faster since there is no need to back-propagate to the early layers of the network.

3.4 Transformation Layers

For our need, it is not enough to obtain good performance on the target dataset. We placed ourselves in a context where the transfer is done at a later time than the original training and the source dataset is not available anymore, but the network

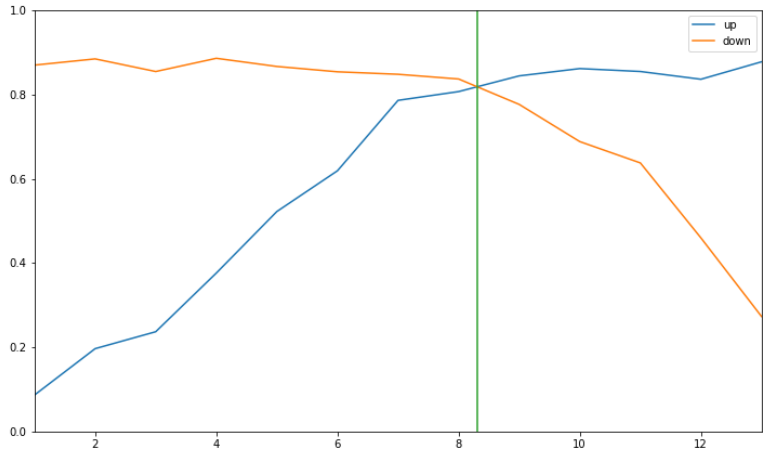


Figure 3.7: Partial fine-tuning. The x axis is the index of the layer and the y axis the Dice index. In blue, all the layers up to x are fine-tuned. In orange, all the layers after x are fine-tuned.

must still be able to segment those images. Because of the performance drop on the source dataset of the fine-tuning method, we must use something else.

Here we propose to introduce *transfer* layers within the original model in order to adapt feature maps to the new dataset. We expect these layers to selectively adapt the network to children data while maintaining the network performance on adults.

When inserting new layers, it might be tempting to simply use standard convolutional layers. The problem is that as the parameters of these layers are fixed once the training is done, the same convolutions will be applied to all images and feature maps. Those new convolutions will heavily degrade performance on images from the original distribution.

We tested many variations of position and shape of those additional convolutional layers. The combination with the highest performance was to add a convolutional layer after each downsampling or upsampling layers, i.e. 6 new layers with filters of size $3 \times 3 \times 3$. This was enough to barely reach the performance of the fine-tuning method, while taking longer and adding 23 millions parameters to the network.

We instead propose to predict parameters during inference for each image. Images from the source distribution will have parameters predicted close to the identity transform, while images from the target distribution can be transformed to be more similar to the source distribution. We introduce two kinds of transfer

layers with different goals.

3.4.1 Geometric Transformation Layer

Working within the same image modality, we propose to use geometric transformations as transfer layers in order to effectively transfer information between our datasets (adults to children). This is similar to the Spatial Transformer proposed by Jaderberg et al. (2015), but instead of predicting directly the parameters of the transformation, we predict the amount of translation, rotation and scaling along each axis, i.e. nine parameters in total. These parameters are then used by the transformation layer to geometrically modify its input using tri-linear interpolation. There is no attention mechanism.

We place this layer at the entrance of the network and perform the inverse transformation at the output of the network as shown in Figure 3.6. The transformation can be seen as a projection on the adults space, and therefore we need to project back on the children space at some point. The prediction of the parameters is done with a 3-layers convolutional network taking the image as input.

3.4.2 Intensity Layer

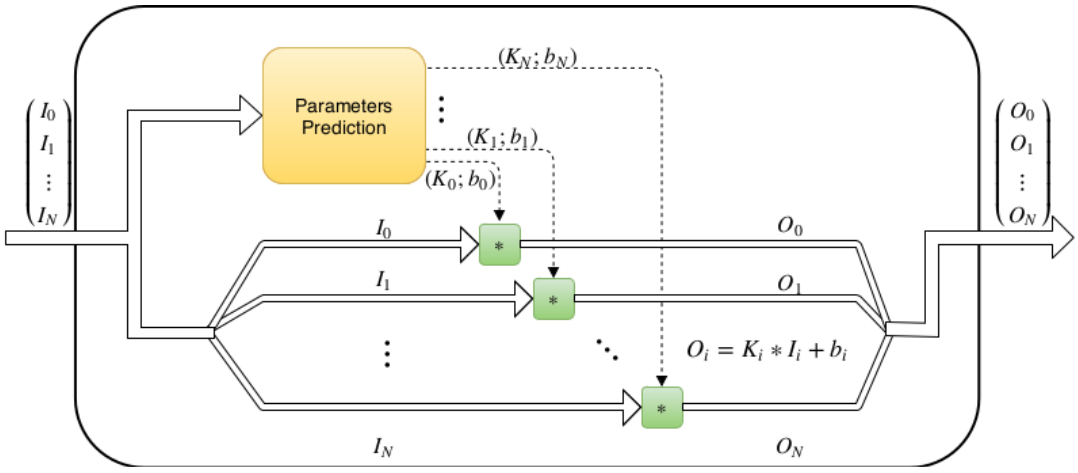


Figure 3.8: Intensity Layer. "Parameters Prediction" can be any model, in our case it is a 3-layers ConvNet.

A geometric transformation is not enough to express the differences between the source and target datasets. We therefore propose to predict an intensity transformation, implemented as a $1 \times 1 \times 1$ convolution applied to each feature map,

shown in Figure 3.8. Noting $I = \{I_1, \cdots, I_n\}$ the input feature maps and θ the weights of the parameters prediction component, the exact transformation applied to each feature map I_i is:

$$O(I_i) = K_i(I;\theta) * I_i + b_i(I;\theta)$$

where $K_i(I;\theta)$ and $b_i(I;\theta)$ are the predicted parameters. In the end, two parameters have to be predicted for each feature map. Experiments showed that this layer works best when placed at the lowest level of the U-Net (see Figure 3.6).

3.5 Results and Discussion

3.5.1 Comparing transfer methods

Strategy	# params	Time (s)	Dice Adults	Dice Children
Adults Baseline	16M	900	**0.80 ± 0.01**	0.61 ± 0.02
Children Baseline	16M	112	0.57 ± 0.03	**0.66 ± 0.02**
Joint Baseline	16M	1790	**0.81 ± 0.01**	**0.74 ± 0.01**
Fine-tuning	0	115	0.72 ± 0.02	0.72 ± 0.03
Fixed Convolutions	23M	182	0.71 ± 0.02	0.72 ± 0.03
Geometric Input (GI)	148k	36	0.73 ± 0.02	0.67 ± 0.02
Intensity	124k	29	0.80 ± 0.01	0.65 ± 0.03
GI + Intensity	272k	37	**0.74 ± 0.01**	**0.73 ± 0.02**

Table 3.1: Performance of the baseline and the different transfer methods. Each result is averaged over 5 random seeds impacting the separation of the database into training and testing sets, and the weights initialization of the networks.

In all transfer experiments, the source network is the adults baseline. The transfer is done only on the children dataset, but performances are evaluated on both the adults and children datasets. Besides fine-tuning, we experimented with the types of layers added to the network and their position as described in Section 3.4. The most interesting results are presented below.

In this section, we show that some of the proposed models achieve good results, both quantitatively as shown in Table 3.1 and qualitatively, as illustrated in Figures 3.11 and 3.10.

Table 3.1 shows the best performing strategies among those we explored. The number of parameters is the number of parameters added by the method to the

base network, and the time is the time it takes for one epoch (in seconds).

For the baseline models, jointly training on adults and children gives better results than training on each class alone, or than any transfer method. This is not surprising as having access to both datasets at the same time is an advantage over our transfer setting where we only have access to one dataset at a time. We conclude that it is the method that should be used if we have access to both datasets at the same time. It can be noted that the performance gap on the children dataset between the adults baseline and the children baseline is smaller than between the children baseline and the joint baseline. In addition, the performance on the adults dataset is better for the joint baseline than the adults baseline. This suggests that the quantity of available data is more important than its quality, i.e. a lot of data from a related distribution is better than a small quantity from the correct distribution.

For the transfer methods, the fine-tuning easily beats the children baseline, but does not reach the performance of the joint baseline on either dataset. Moreover, while it does not add any parameter to the network, it takes more time to train than most other transfer methods (this is due to the fact that we have to compute the gradient of every parameter in the network).

Adding fixed convolutional layers did not work. We were able to reach the performance of the fine tuning, but only by adding $3 \times 3 \times 3$ convolutional layers after every downsampling and upsampling layers (line "FIxed Convolutions" in Table 3.1). This added 23 millions parameters to the network (it originally had 16 millions) and took longer to train than the fine tuning.

"Geometric Input" means that we predict the parameters of a geometric transformation before the first layer of the network as described in Section 3.4.1. It is fast to train and beats the children baseline but fine tuning gives better performance. Likewise for "Intensity" only, which means putting an intensity layer at the deepest layer of the U-Net (Section 3.4.2).

The best results were obtained by combining the geometric input and learning to predict the parameters of an intensity transformation of the feature maps at the lowest level of the U-Net. This position for the intensity transformation works better than the others. This method adds a small amount of parameters, is fast to train and improve the performance over fine tuning.

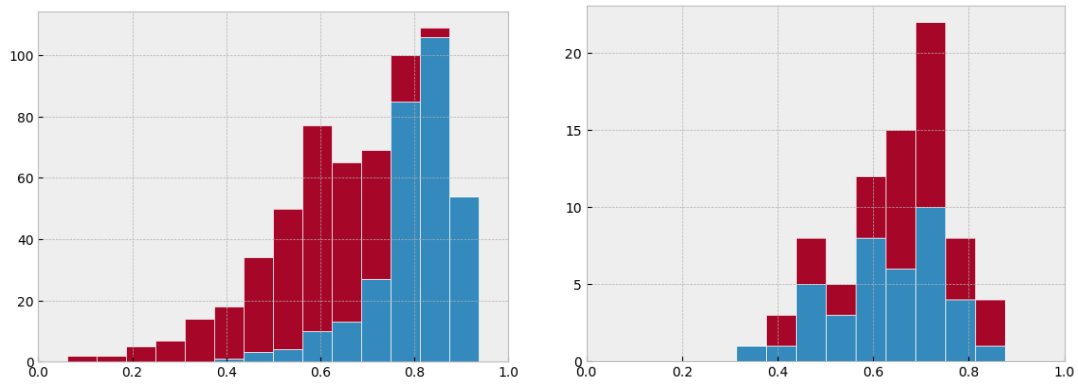


Figure 3.9: Histogram of Dice coefficient on baseline models. In red is the adults baseline, blue is the children baseline. Left: Dice coefficient on adults, right: Dice coefficient on children.

3.5.2 Examining common failures

Examining in details the results of the different networks, we can observe a few trends. From the histograms of the Dice coefficient in Figure 3.9, we note that a few of them have a very low Dice index (< 0.20). We show some of these images in Figure 3.11. Moreover they are consistently bad for all tested models. All three images are of very low quality, and there are shadows hiding parts of the images. Fortunately, they represent less than 1% of our dataset.

Looking at the other images, two kinds of failures were dominant. Either there were multiple connected component in the segmentation or the segmentation were bigger than the ground truth. Some representative samples are shown in Figure 3.10. The multiple connected component in the segmentation result from a lack of constraint on the shape of the segmentation. A big segmentation containing the ground truth or a segmentation with multiple connected component can have a high dice value. Adding constraints to the loss could also help to solve some of the common mistakes. Other ways to alleviate the problem would be to smooth the segmentation or to use a shape model, for example using a Conditional Random Field such as in Kamnitsas, Ledig, et al. (2017).

3.6 Conclusion

We proposed a new transfer method taking advantage of the particularity of the image format of our problem to obtain better results than the more generic fine-

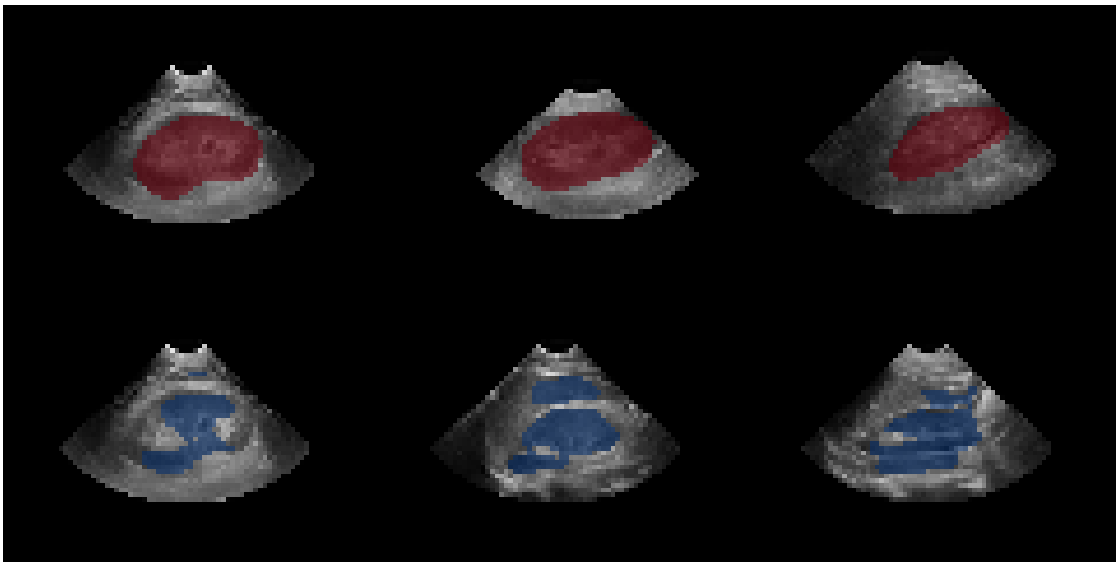


Figure 3.10: Middle slice of 6 different volumes and a network segmentation superposed on it. The top row shows good segmentations with a Dice index > 0.90. The bottom row shows segmentations with a Dice index between 0.60 and 0.70. They show the most common mistakes made by the network.

tuning. This new method could be applied to other medical imaging problems, with a few caveats. First, if access to the source dataset is possible, joint training on the source and target should be preferred over any transfer method. Secondly, if changes to the structure of the network are impossible (for example if the architecture is hard-coded in released software), our method cannot be used. Finally, it depends on the nature of the differences between the source and target distributions. Our method is efficient in the case of geometric and/or intensity differences, making it well suited for medical imaging problems.

Figure 3.11: Images on which all models fail to generalize well.

4 Template Deformation and Deep Learning

Abstract

This chapter presents a new segmentation method that combines deep learning and template deformation. Section 4.2 briefly discusses template construction strategies and registration via deep learning. The *implicit template deformation* framework on which this work is based, is introduced in Section 4.3. Our method is described in Section 4.4 and tested on the task of kidney segmentation in 3D ultrasound images (Section 4.5). The results are discussed in Section 4.6.

Contents

4.1 Introduction

In medical imaging, the use of prior knowledge can greatly facilitate the segmentation of anatomical structures. This can be done by constraining the segmentation to be close to a pre-defined shape, i.e. a shape prior. Despite their previous popularity, using a shape prior is uncommon in modern deep learning, where the standard method is to produce the segmentation only from the image.

Template deformation produces the segmentation by deforming a binary template to match the image. Two components are needed: a template (the shape prior) and a regularization imposed on the deformation (the shape constraint). In this work we investigate how deep learning can be used for template deformation, and in particular how it can improve the *implicit template deformation* framework. The core idea is to use the network to predict the transformation to be applied to the template.

Using deep learning for this task has two advantages: (1) because the network is trained on a database of images and does not have to be retrained on new images, the segmentation is extremely fast (even if using only CPU during inference); (2) the loss function only requires the ground-truth segmentation, there is no need to have the ground-truth of the geometric deformation or additional information.

As this work was done at the very end of the PhD, the results are still preliminary, yet promising.

4.2 Template Deformation

An approach to the segmentation of medical images is to use a previously acquired template and deform it to match the image to segment. Formally, given a template ϕ_0 and an image I, we are interested in obtaining the segmentation mask ϕ by finding the transformation ψ to be applied to the template:

$$\phi = \phi_0 \circ \psi \tag{4.1}$$

4.2.1 Constructing the template

The template can take different forms and there are various strategies to construct it. One can select the template as ground-truth segmentation of the most similar image to the one to segment in an existing database (Commowick and Malandain (2007)) or build a mean template by using multiple images (S. Joshi et al. (2004)).

Another strategy is to use multiple templates, which increases the robustness of the methods (Heckemann et al. (2006)) and then fuse the predictions (Warfield, Zou, and Wells (2004)).

A review of the existing methods of template construction can be found in- Cabezas et al. (2011).

4.2.2 Finding the transformation

There is a vast literature of methods such as *Active Shape Models* (T. Cootes et al. (1995)), *Active Appearance Models* (T. F. Cootes, Edwards, and Taylor (1998)) or *Implicit Template Deformation* (Saddi et al. (2007)) that have been proposed to find the deformation to apply to the template. These methods make no use of machine learning, and instead work by minimizing some kind of energy functional. We refer the interested reader to Heimann and Meinzer (2009) for a review of these methods.

To the best of our knowledge, deep learning has not yet been used in the context of template models. It has however been used in the context of registration, i.e. the spatial alignment of two medical images.

Convolutional neural networks have been used recently to regress the parameters of the registration transformation from the input images (Miao, Wang, and Liao (2016), X. Yang, Kwitt, and Niethammer (2016)).

Another approach is to estimate a similarity measure from a neural network to be used in an iterative optimization strategy (Wu et al. (2013), Cheng, Zhang, and Zheng (2015), Simonovsky et al. (2016)).

Recently, method using GANs (Goodfellow, Pouget-Abadie, et al. (2014)) have been proposed. Dalca et al. (2018) developed a diffeomorphic integration layer coupled with a generative model that allows for registration without the ground-truth deformation fields. In the approach of Fan et al. (2018), a registration network predicts the deformation and a discrimination network evaluates the quality of the alignment. The networks are trained in the typical GAN fashion. A regularization term is added to the loss of the registration network to make the deformation field smoother.

4.3 Segmentation by Implicit Template Deformation

As this work places itself on top of Mory et al. (2012) and Prevost (2013), we first present the *Implicit Template Deformation* framework.

The segmentation is defined as the zero level-set of an implicit function $\phi : \Omega \to \mathbb{R}$ where ϕ is positive inside the segmentation and negative outside. The set of admissible segmentations $\mathbb{S}$ is defined as the set of all implicit functions with the same topology as the implicit template ϕ_0:

$$\mathbb{S} = \{\phi : \Omega \to \mathbb{R} \quad \text{s.t. } \phi = \phi_0 \circ \psi,\ \psi \text{ is diffeomorphic}\} \tag{4.2}$$

Let H denote the Heaviside function ($H(a) = 1$ if $a > 0$, 0 otherwise), and r_{int} and r_{ext} image-based functions such that $r_{int}(x) - r_{ext}$ is negative if the pixel x belongs to the target object, positive otherwise. The implicit template deformation aims to find the transformation $\psi : \Omega \to \Omega$ that minimizes the energy:

$$\min_{\psi} \left\{ \int_{\Omega} H\left(\phi_0 \circ \psi\right) r_{int} + \int_{\Omega} \left(1 - H\left(\phi_0 \circ \psi\right)\right) r_{ext} + \lambda \mathbf{R}\left(\psi\right) \right\} \tag{4.3}$$

where $\mathbf{R}\left(\psi\right)$ is the regularization term that prevents the segmentation $\phi = \phi_0 \circ \psi$ to deviate too much from the initial template ϕ_0.

In the work of Mory et al. (2012), the transformation ψ is the composition of a global transformation G and a local transformation L:

$$\psi = L \circ G \tag{4.4}$$

The global transformation G is a parametric transform that globally aligns the template with the target. It is usually a similarity, as it is adapted to anatomical structures.

The local transformation is defined by a displacement field u in the template referential $L = u + Id$, where u is the smoothed version of a displacement field v with a Gaussian filter K_σ:

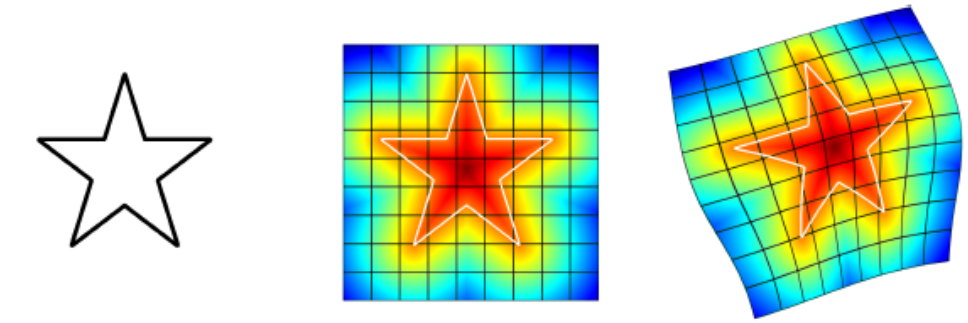
$$u(x) = [K_\sigma * v](x) = \int_{\Omega} K_\sigma(x - y) v(y) dy \tag{4.5}$$

Filtering the field with a Gaussian kernel enforces its regularity at a low computational cost.

The idea of separating the transformation into a global and local component comes from Yezzi and Soatto (2003). The global transformation modifies the *pose* of the prior while preserving its shape, which is changed by the local transformation.

The advantage of this decomposition is to be able to define the regularization term $\mathbf{R}(\psi)$ independently from the pose. The constraint $\mathbf{R}$ then controls only the deviation of the shape of the segmentation ϕ from its prior ϕ_0. It is chosen as an L_2 norm to constrain L towards the identity Id and the magnitude of the deformation:

$$\mathbf{R}(\psi) = \mathbf{R}(L) = \frac{\lambda}{2}||L - Id||_2^2 \tag{4.6}$$



(a) star-shaped prior (b) implicit template ϕ_0 (c) deformed $\phi = \phi_0 \circ \psi$

Figure 4.1: Implicit template deformation: the segmentation corresponds to the deformed initial template. (Source: Mory (2011))

Once the deformation has been found through the minimization of Equation 4.3, typically through a gradient descent scheme to update G and L from a carefully chosen initialization, the segmentation is obtained by applying it to the template, as illustrated in Figure 4.1.

4.4 Template Deformation via Deep Learning

The method presented in this section produces the segmentation of an image given this image and a fixed template. The deformation to be applied to the template is predicted by a neural network. Following the reasoning of Mory et al. (2012), we define the deformation ψ as the composition of a global transformation G and a local transformation L: $\psi = L \circ G$.

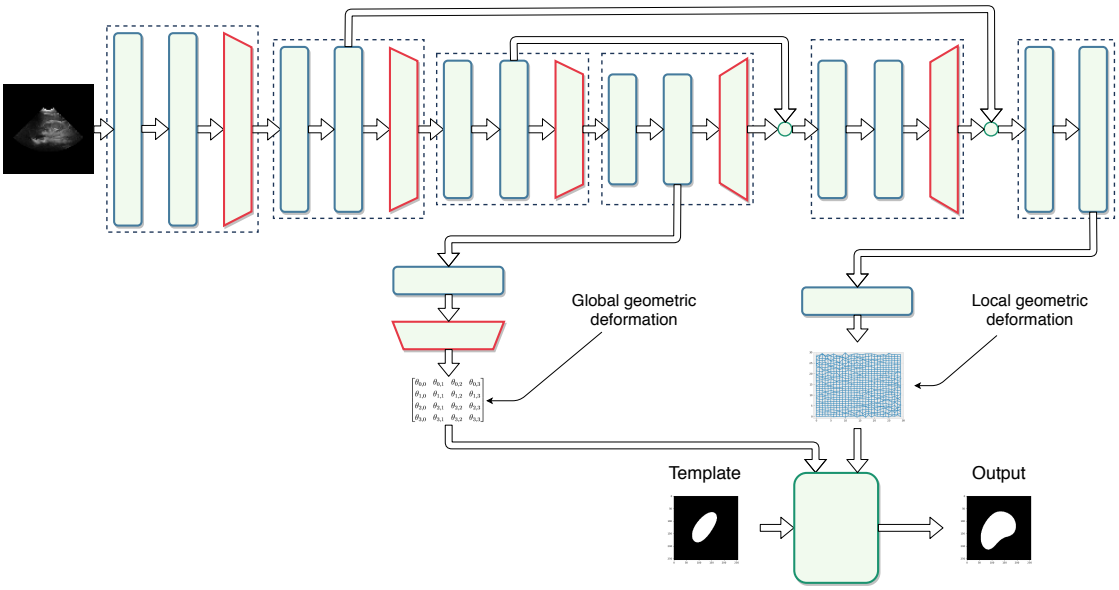


Figure 4.2: Segmentation network that predicts a global geometric transformation and a deformation field to deform a template.

Figure 4.2 shows the neural network. The network predicts the global transformation G and the local transformation L from the image, which are then applied to the template ϕ_0:

$$\phi = \phi_0 \circ \ L \circ G \tag{4.7}$$

The network architecture for the local transformation is very similar to a U-Net, as the transformation must be produced at a relatively high resolution and the U-Net aggregates information from many levels. The global transformation is predicted by a classical ConvNet. As the tasks are closely related, the classical ConvNet corresponds to the downward part of the U-Net, allowing for transfer of information.

The next sections detail our choice of global transformation (Section 4.4.1), of local transformation (Section 4.4.2) and how to train the network (Section 4.4.3).

4.4.1 Global transformation

A linear geometric transformation for 3D images is represented by a 4×4 matrix, meaning a total of 16 parameters that can be predicted. Directly predicting those parameters remove control on the kind of transformation that are predicted. In

our case, we chose to predict instead translation, rotation and scaling parameters.

Three parameters are predicted for the translation on each axis, giving the following translation matrix T:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The scaling matrix S is built from three more parameters:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, we have one rotation matrix in each direction (R_x, R_y and R_z) built from one parameter each:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos r_x & -\sin r_x & 0 \\ 0 & \sin r_x & \cos r_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos r_y & 0 & -\sin r_y & 0 \\ 0 & 1 & 0 & 0 \\ \sin r_y & 0 & \cos r_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z = \begin{bmatrix} \cos r_z & -\sin r_z & 0 & 0 \\ \sin r_z & \cos r_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We also need to center the image at the origin before applying the rotations, which requires no parameters except knowing the center of the image:

$$C_+ = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 0 & 0 & c_y \\ 0 & 0 & 0 & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad C_- = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 0 & 0 & -c_y \\ 0 & 0 & 0 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From these matrices, the geometric transformation G applied to the shape model

is the following:

$$G = T \cdot C_+ \cdot R_z \cdot R_y \cdot R_x \cdot S \cdot C_- \tag{4.8}$$

The prediction of the 9 required parameters is done by a convolutional layer with 9 $1 \times 1 \times 1$ filters, followed by an average pooling layer of the size of the feature maps.

4.4.2 Local deformation

The deformation field v is predicted from a convolutional layer with 3 $3 \times 3 \times 3$ filters, one filter per dimension. Each field is then smoothed with a $3 \times 3 \times 3$ mean filter, before being resized to the shape model size with a tri-linear interpolation. This resizing step allows predicting the deformation field at a lower resolution than the shape model, saving time and parameters to learn.

As mentioned in Section 4.3, the local deformation is then computed from v as:

$$L = u(x) + Id = [K_\sigma * v](x) + Id \tag{4.9}$$

As in the implicit template deformation case, the Gaussian filtering is used here to enforce the regularity of the deformation field.

4.4.3 Training and loss function

Our segmentation method requires only the usual database of images and their ground-truth segmentations expressed as binary masks. For simplicity (and lack of time), the template is chosen as one of these segmentations. The corresponding image is then excluded from training and testing.

Applying the global and local transformations to the template can be done as a layer of the network that outputs the final segmentation. The loss function J is simply the Dice coefficient between the network output $\hat{\phi}$ and the ground-truth segmentation ϕ:

$$J\left(\phi, \hat{\phi}\right) = DICE\left(\phi, \hat{\phi}\right) = DICE\left(\phi_0 \circ L \circ G, \hat{\phi}\right) \tag{4.10}$$

The only free variables are G and L, which are predicted by the network. Training the network by backpropagation on a database of images allows it to predict G and L for new images.

4.5 Kidney Segmentation in 3D Ultrasound

The problem addressed in this section is the same as in Section 3.2: kidney capsule segmentation in 3D ultrasound data of potentially ill children. The difference is that we are not in a transfer learning setting and we have access to both adults and children images simultaneously.

While our new segmentation method is applicable to any kind of medical imaging segmentation problem, time constraint forced us to focus on an already available dataset. As the problem is difficult (in particular, US acquisitions are very noisy), this is still a good test of the method, though further testing is required.

The dataset stays identical as in Section 3.2.3, as well as the pre-processing. We have 503 healthy adults images and 64 children. The images have been downsampled to a resolution of $4 \times 4 \times 4$ mm^3, and centered in a $80 \times 80 \times 80$ voxels cube.

	Mean	Standard deviation
Gaussian noise	0	5
Translation	0	3
Rotation	0	0.05
Scaling	1	0.05

Table 4.1: Data augmentation used for the kidney dataset. The parameters for each type of augmentation are drawn from a normal distribution with the specified mean and standard deviation.

Unlike previously, we use data augmentation. Due to the already high cost of training on this dataset, each image is augmented 10 times before training. The augmentation includes adding Gaussian noise, as well as translation, rotation and scaling of the image (and the segmentation). The range of each type of augmentation is shown in Table 4.1.

4.6 Results and Discussion

For all the models the training is done jointly on adults and children with oversampling of the children to balance the populations. Each model was trained on only one seed due to the high cost of training (around five days per model on a NVIDIA TITAN X). The performance of each model in terms of Dice index is reported in Table 4.2.

Method	Dice Adults	Dice Children
3D U-Net (no data augmentation)	0.81	0.74
3D U-Net	**0.89**	**0.82**
Deformation	0.85	0.75
Geometric transfo + Deformation	0.88	**0.82**

Table 4.2: Performance of the baseline and the shape model methods.

The baseline for our comparison is the 3D U-Net (Çiçek et al. (2016)) described in Section 3.3.1. Its performance is therefore the same as the one reported in Section 3.5. The use of data augmentation results in a performance gain from 0.81 for the adults and 0.74 for the children to 0.89 and 0.82, an important increase of 0.08 for both populations.

Training a model predicting only deformation fields without the geometric transformation is not enough to beat the baseline, though it does show that the idea works. When adding the geometric transformation, the results become very close to the baseline. The geometric transformation is beneficial in particular for the children. This is likely due to the fact that the template model is an adult kidney, requiring the network to learn how to shrink it when segmenting children images. The geometric transformation provides an easy way to learn this shrinking with the three scaling parameters.

Even though the performance of our method is similar to the baseline, using the template model results in anatomically correct segmentation. There are no patches of disconnected voxels classified as kidney or unlikely shapes. In Figure 4.3, we selected three images from the test set to illustrate this. In the first one, the U-Net segmented a patch of pixel in the top left disconnected from the rest[1], while our method did not. The second image is difficult as the huge black spot can be mistaken for hydronephrosis. While both networks classified it as kidney, our method kept the elongated shape of the kidney outside of the black spot. In the third image, the U-Net segmented a patch of pixels almost disconnected from the main patch which would be very difficult to obtain by deforming a template model, as our method shows.

Since the geometric transformation provided an important boost in performance, we look at the distribution of each parameter predicted on the test set in Figure 4.4.

[1] As we show only a slice of a volume, the patch could be connected through the other slices, this is not the case here.

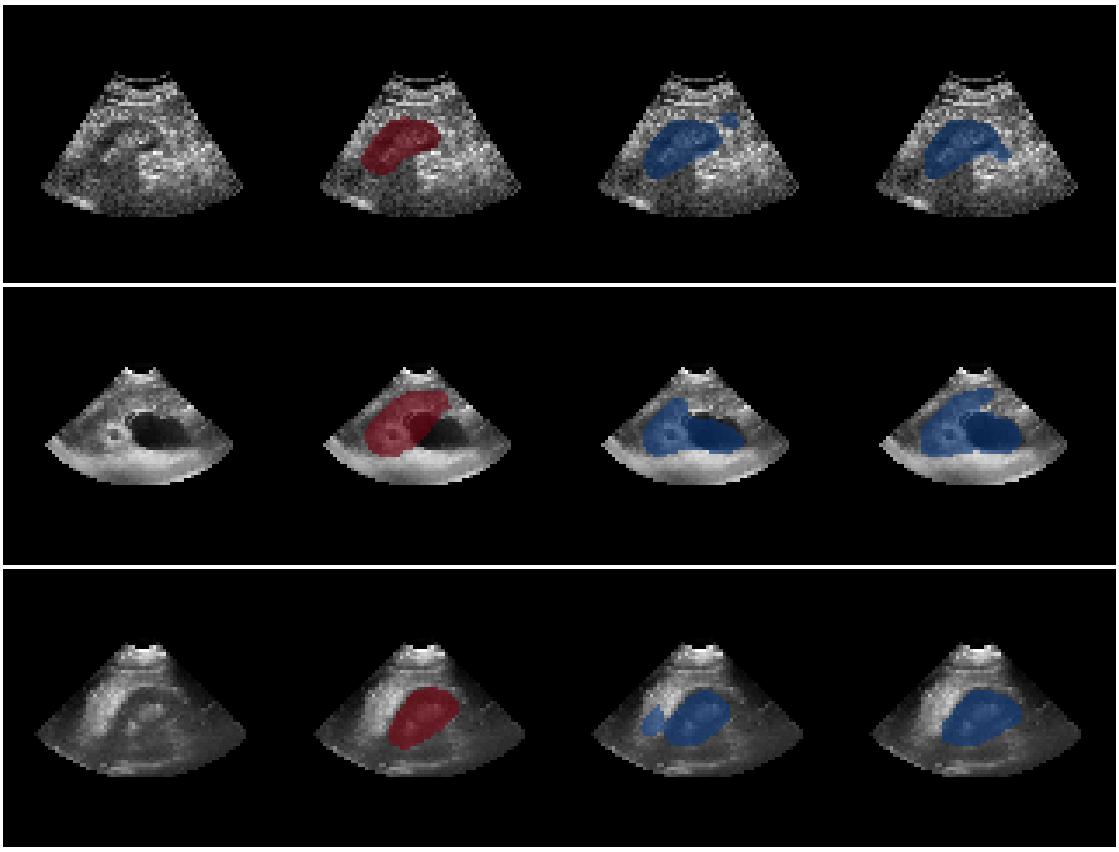


Figure 4.3: Comparing 3D U-Net results and deformation fields results. From left to right: image, ground truth, U-Net segmentation, deformation fields segmentation.

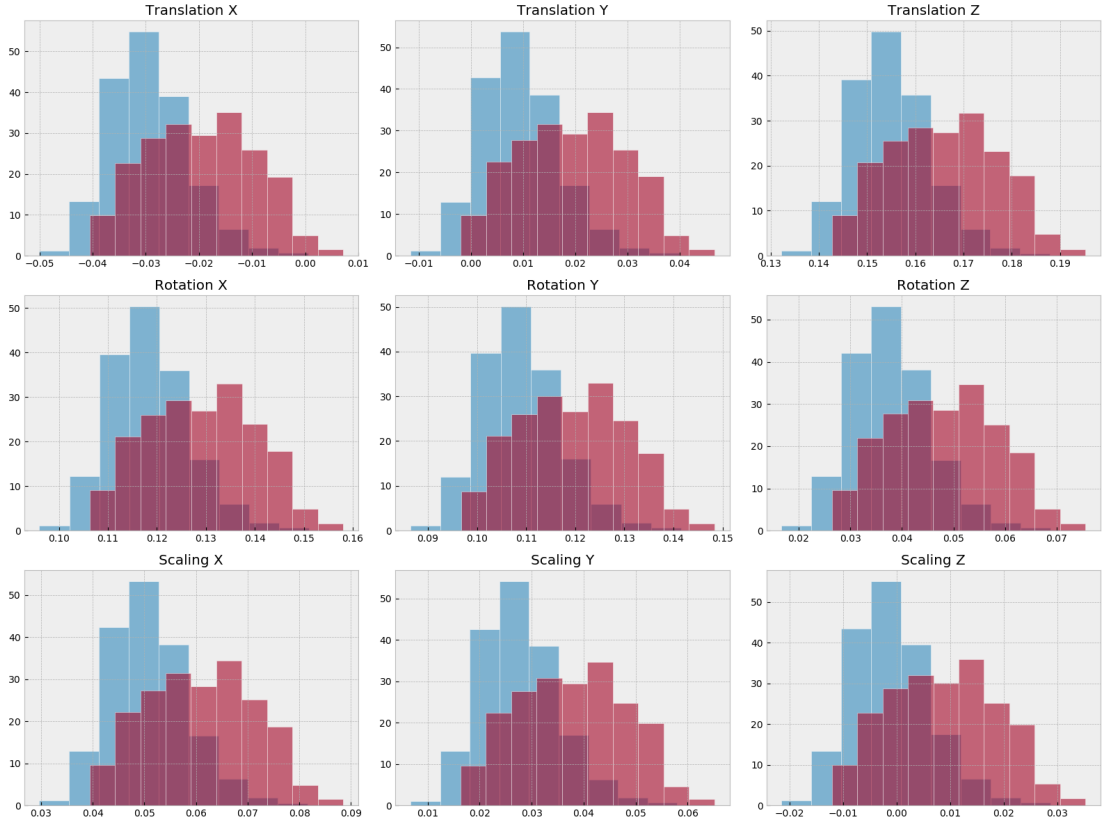


Figure 4.4: Distributions of each parameter of the geometric transformation for the final model. Red is for the adults, blue for the children.

First we note that the distributions are almost identical for all parameters. Looking at the values for individual images reveals that the parameters are correlated, i.e. if a parameter falls on the lower side of the distribution for an image, the other parameters will also fall on the lower side of their distributions for the same image. This is likely due to the use of only one convolutional layer to predict the parameters, i.e. a lack of capacity.

The second point of interest is that the distributions for adults and for children are very different. The children tend to have lower parameter values and the adults are spread out over a bigger range. This makes sense for the scaling parameters: as the template is based on an adult's kidney, it is necessary to shrink it more to match the size of a children. However there are no reasons why this should be true for the translation and rotation. This is possibly a side-effect of the parameters being so correlated. If the network lacked capacity and some parameters are much more important than the others, it makes sense it would focus on them, in this case the scaling, at the detriment of the others, here the translation and rotation.

Finally, looking at the actual value of the parameters, it seems that the network relies on a very specific transformation to work. Every parameter falls into a narrow range of values. For all images, the template is translated, rotated and scaled in roughly the same direction and amplitude.

The very low values of the scaling means that the template model is heavily shrunk. As a result, the deformation fields have very high values to match the target.

4.7 Conclusion

In this chapter we presented a new segmentation method based on template deformation via deep learning. This method does not require ground-truth deformation fields, but only the standard ground-truth segmentation masks. As such, it can be applied to any medical imaging segmentation problem where the use of a template makes sense.

As this work was done at the very end of the PhD, it is still incomplete, and there are some simple modifications that would likely improve the performance.

To avoid the very low values of the scaling parameters, one solution is to add an L_2 regularization term to the deformation field L:

$$\mathbf{R}(L) = \frac{\lambda}{2}||L - Id||_2^2 \tag{4.11}$$

This would strongly reduce the magnitude of the deformation field and force the scaling into a more sensible range.

To counter the strong correlation among the parameters of the global transformation, we should increase the capacity of the network by adding one or two convolutional layers just before the parameters prediction.

The other aspect of this work that merits attention is the choice of template. We chose the simplest method of simply using one of the ground-truth segmentations as the template, but more complex methods are likely to yield better results. In particular the work of Prevost (2013) learns the mean shape and possible variations and integrates them into the implicit template deformation framework, making it a natural fit with our approach.

5 Conclusion

5.1 Summary of the contributions

When this thesis began in early 2016, deep learning had already shown its worth on natural images, but contributions in medical imaging were rare. There was still a strong lack of tools and understanding on how to build architectures adapted to a specific problem, which naturally led us to the topic of hyper-parameter optimization, in order to avoid tedious architecture handcrafting and hyper-parameter fine tuning.

Now armed with a set of tools to quickly find good models, the second part of this thesis focused on applications. Questions of transfer learning and template deformation, and their link with deep learning were explored in this context. They came from the lack of data so common in medical imaging, and thus a need to re-use the limited knowledge at our disposal as much as possible.

Hyper-parameter optimization

- **An incremental Cholesky decomposition to reduce the cost of Bayesian optimization.** Most of the computational cost of Bayesian optimization is in the inversion of the Gaussian process' Gram matrix. We exploited a specificity in the structure of this matrix in the case of Bayesian optimization: each successive call adds new rows and columns while leaving the rest of the matrix unchanged. We have shown that this property stays true for the underlying Cholesky decomposition, and how to compute the new decomposition faster when the previous decomposition is available.

- **A limited comparison of the performance of random search and Bayesian optimization.** We designed an experiment on a small hyper-parameter space to observe the behavior of random search and Bayesian optimization over many runs. Bayesian optimization found faster better models than random search in the best, average and worst cases. We showed that the Gaussian process quickly became a good predictor of model performance and that the worst models were picked last. Random search behaved in accordance with the theoretical bounds we derived.

- **A new hyper-parameter optimization method combining Hyperband and Bayesian optimization.** We proposed a method combining the strengths of Hyperband and Bayesian optimization. Model selection is done by Bayesian optimization, and model training follows the Hyperband scheme. Unfortunately due to how the selection of multiple models simultaneously was handled, the method did not perform significantly better than Hyperband alone.

A method to solve a classification problem of MRI field-of-view. Using a dataset of MRI volumes from a multitude of protocols and machines, we developed a neural network able to classify each slice of the volumes into their anatomical regions (six classes such as head or pelvis). We improved this neural network by using Bayesian optimization to find a better architecture providing a non-negligible performance boost. Even though the classification was done at the slice level, we showed that it could be used for robust region localization through a decision scheme maximizing the likelihood of each region.

A new transfer learning method and its application to the segmentation of the kidney in 3D ultrasound images. Working with a dataset of 3D ultrasound kidney images across two populations, we investigated transfer learning methods for the segmentation of the kidney from one population (healthy adults) to the other population (sick children), where less examples are available. This led us to develop a new transfer learning approach, based on adding layers to the pre-trained network to predict parameters for geometric and intensity transformations.

A segmentation method of template deformation using deep learning. The use of shape prior is still uncommon in deep learning. We proposed a new segmentation method based on the *implicit template deformation* framework that

uses deep learning to predict the transformation to be applied to the template. While this work is still preliminary, we obtained competitive performance on the 3D US kidney segmentation task previously explored.

5.2 Future Work

Even though all the contributions listed in the previous section have immediate ways they could be improved, those are discussed in their respective chapters. Here we take a longer term perspective and suggest future research directions, taking into account how we believe the field is going to evolve.

User interactions. An important aspect of the implicit template deformation framework is that it can integrate user interactions. After the template is deformed, a user can add or remove points from the segmentation. Those are taken into account by the method, which produces a new refined segmentation. This process can be repeated as many times as required. This kind of user interaction is key in medical practice but deep learning provides no easy way to incorporate them. While some attempts have been made such as by Çiçek et al. (2016), who provide a partial segmentation as an additional input of the segmentation network, none of them forces the network to make use of them. In our method of template deformation, user input could be incorporated as additional constraints on the deformation field, while forcing the final segmentation to still be an acceptable shape.

A shift in applications. Deep learning has fulfilled one of its promises: many of the tasks that were once difficult are now easily solvable with deep learning. The classification task of MRI field-of-view we worked on is a good example. Once the dataset is ready, automated methods of hyper-parameter optimization can be used to find a very efficient model with little manual effort. As automation tools improves and becomes more accessible[1], many tasks will stop requiring image processing expertise to be solved. The focus will move to more challenging tasks, where the difficulty can come from a lack of data (rare disease, high variability), greater complexity such as segmentation of small structures (due to the reliance of neural networks on pooling layers), registration, surgical planning, ...

[1] See for example Google AutoML.

An integration of older methods. This thesis first started completely focused on deep leaning and how it could be used to solve medical imaging problems. By the end of it, we had started working on a hybrid approach of deep learning and template deformation. Pre-deep learning methods were tailored for medical tasks, and were mostly discarded with the success of deep learning. But deep learning was developed first for natural images applications, without accounting for the differences between natural and medical images. The ideas used in these older methods (such as the use of shape information) are still relevant, and we expect their integration with deep learning will move the field forward in coming years.

Multi-modality, multi-task learning. The idea of transfer learning is to share knowledge between more or less loosely connected problems. Expanding on this, we could imagine a multi-input multi-output model that shares knowledge over many tasks and modalities at the same time. A first step in this direction could be the construction of modality-specific pre-trained models, a medical imaging equivalent to ImageNet pre-trained models. This requires that huge datasets are available per modality, but efforts in this direction are ongoing. For example, the NIH recently released an X-Ray dataset of over $100,000$ images that could be used to build a standard X-Ray pre-trained model. If ImageNet pre-trained models can improve performance on most medical tasks, it seems likely that a more specific model would improve performance even more.

Differential privacy. Due to the sensitive nature of medical images, sharing between research institutes or hospitals is difficult. A solution to this problem could be through the use of differential privacy. It is a mathematical framework in which the output of an algorithm is mostly the same, whether the data from one particular patient is included or not. This prevents an opponent from recovering patient information from the algorithm (it is possible to reconstruct training set images from computer vision systems, including deep neural networks, see for example Fredrikson, Jha, and Ristenpart (2015)). There has been some work to make deep neural networks differentially private (Martin Abadi et al. (2016)), and this would allow institutions to release models trained on private data with strong guarantees of patient privacy.

A

Incremental Cholesky Decomposition Proofs

This appendix contains the derivations for obtaining the incremental Cholesky decomposition and its inverse that are used in Section 2.3. For recall, the Cholesky decomposition and its inverse can be decomposed into blocks where one is the previous Cholesky decomposition. We obtain the formulas by developing this block decomposition.

Let us recall that the Cholesky decomposition $L_{(n)}$ of a positive definite matrix $K_{(n)}$ is a decomposition of the form:

$$K_{(n)} = L_{(n)} L_{(n)}^T \tag{A.1}$$

where $L_{(n)}$ is a lower triangular matrix. The decomposition is unique only if $K_{(n)}$ is positive definite. Since $K_{(n)}$ is a Gram matrix, it is always guaranteed to be positive semi-definite (i.e. $\forall v \in \mathbb{R}^n, \;\; v^T K_{(n)} v \geq 0$). On top of that, if the rows and columns are unique (i.e. there are no duplicated data points), then it is positive definite.

A.1 Formula for $L_{(n+k)}$

When adding k points to a Gram matrix of n points, the block decomposition of the new Cholesky decomposition is:

$$\begin{aligned} & L_{(n+k)} L_{(n+k)}^T = K_{(n+k,n+k)} \\ \Leftrightarrow \quad & \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} A^T & C^T \\ B^T & D^T \end{pmatrix} = \begin{pmatrix} K_{(n,n)} & K_{(k,n)}^T \\ K_{(k,n)} & K_{(k,k)} \end{pmatrix} \end{aligned}$$

B is obviously 0 since a Cholesky decomposition is lower triangular.

By developing the block decomposition, we obtain the following equation for A:

$$AA^T + BB^T = AA^T = K_{(n,n)}$$

Since the Cholesky decomposition is unique, $A = L_{(n)}$. The equation for C is also simple to solve:

$$\begin{aligned} & CA^T + DB^T = CL_{(n)}^T = K_{(k,n)} \\ \Leftrightarrow \quad & C = K_{(k,n)}(L_{(n)}^T)^{-1} \end{aligned}$$

Solving for D requires more work. Developing the block decomposition, we have:

$$\begin{aligned} & CC^T + DD^T = K_{(k,k)} \\ \Leftrightarrow \quad & DD^T = K_{(k,k)} - K_{(k,n)}(K_{(n,n)})^{-1}K_{(k,n)}^T \end{aligned}$$

If we can show that $K_{(k,k)} - K_{(k,n)}(K_{(n,n)})^{-1}K_{(k,n)}^T$ is definite positive then D is its unique Cholesky decomposition.

Let $w \in \mathbb{R}^{n+k}$ s.t. $w = \begin{pmatrix} u \\ v \end{pmatrix}, u \in \mathbb{R}^n, v \in \mathbb{R}^k$, we have $w^T K_{(n+k,n+k)} w \geq 0$ because $K_{(n+k,n+k)}$ is a Gram matrix. Using its block decomposition and developing it, we have:

$$\begin{aligned} & \begin{pmatrix} u \\ v \end{pmatrix}^T \begin{pmatrix} K_{(n,n)} & K_{(k,n)}^T \\ K_{(k,n)} & K_{(k,k)} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \geq 0 \\ \Leftrightarrow \quad & u^T K_{(n,n)} u + u^T K_{(k,n)}^T v + v^T K_{(k,n)} u + v^T K_{(k,k)} v \geq 0 \\ \Leftrightarrow \quad & u^T K_{(n,n)} u + 2u^T K_{(k,n)}^T v + v^T K_{(k,k)} v \geq 0 \end{aligned}$$

Using three successive change of variables, this equation becomes a second-order polynomial. The first change is $u = K_{(n,n)}^{-1} x$:

$$x^T (K_{(n,n)}^T)^{-1} x + 2x^T (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v + v^T K_{(k,k)} v \geq 0$$

The second change is $x = ty$ where t is a scalar:

$$t^2 y^T (K_{(n,n)}^T)^{-1} y + 2ty^T (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v + v^T K_{(k,k)} v \geq 0$$

Since this is true for all $y \in \mathbb{R}^n$, this is true in particular when $y = K_{(k,n)}^T v$:

$$t^2 v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v + 2t v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v + v^T K_{(k,k)} v \geq 0$$

The discriminant of this polynomial is negative or null because the polynomial is always positive or null:

$$\begin{aligned} & 4(v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v)^2 - 4(v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v)(v^T K_{(k,k)} v) \leq 0 \\ \Leftrightarrow \quad & 0 \leq (v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v)^2 \leq (v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v)(v^T K_{(k,k)} v) \end{aligned}$$

$v^T K_{(k,k)} v \geq 0$ since $K_{(k,k)}$ is a Gram matrix (i.e. positive semi-definite), meaning by necessity $v^T K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T v \geq 0$ and $K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T$ is positive semi-definite.

Moreover, $K_{(k,k)} \geq K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T$, allowing us to conclude that $K_{(k,k)} - K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T$ is positive semi-definite, and it is positive definite as long as none of the k new combinations are duplicates of the n previous combinations and DD^T is its Cholesky decomposition.

$$D = cho(K_{(k,k)} - K_{(k,n)} (K_{(n,n)}^T)^{-1} K_{(k,n)}^T) = L_{(k)}$$

The final formula for the incremental Cholesky decomposition is:

$$L_{(n+k)} = \begin{pmatrix} L_{(n)} & 0 \\ K_{(k,n)} (L_{(n)}^T)^{-1} & L_{(k)} \end{pmatrix} \tag{A.2}$$

Since this formula requires the costly computation of $L_{(n)}^{-1}$, we would also like to find an incremental formula for it.

A.2 Formula for $L_{(n+k)}^{-1}$

A standard expression for the inversion of a block matrix is:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{pmatrix}$$

Simplifying with $L_{(n+k)}$ found previously, we have:

$$L_{(n+k)}^{-1} = \begin{pmatrix} L_{(n)} & 0 \\ K_{(k,n)}(L_{(n)}^T)^{-1} & L_{(k)} \end{pmatrix}^{-1} = \begin{pmatrix} L_{(n)}^{-1} & 0 \\ -L_{(k)}^{-1}K_{(k,n)}(L_{(n)}^T)^{-1}L_{(n)}^{-1} & L_{(k)}^{-1} \end{pmatrix} \quad \text{(A.3)}$$

Publications

This thesis resulted in the following publications.

Bertrand, Hadrien, Roberto Ardon, et al. (2017). "Hyperparameter Optimization of Deep Neural Networks: Combining Hyperband with Bayesian Model Selection". *Conférence sur l'Apprentissage automatique (CAp)*.

Bertrand, Hadrien, Matthieu Perrot, et al. (2017). "Classification of MRI data using deep learning and Gaussian process-based model selection". *IEEE 14th International Symposium on Biomedical Imaging (ISBI)*.

Bibliography

Each header is a clickable link to the publication.

Abadi, Martin et al. (2016). "Deep Learning with Differential Privacy". *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*.

Angeline, P. J., G. M. Saunders, and J. B. Pollack (1994). "An evolutionary algorithm that constructs recurrent neural networks". *IEEE Transactions on Neural Networks*.

Argyriou, Andreas, Theodoros Evgeniou, and Massimiliano Pontil (2006). "Multi-task Feature Learning". *International Conference on Neural Information Processing Systems*.

Arnold, A., R. Nallapati, and W. W. Cohen (2007). "A Comparative Study of Methods for Transductive Transfer Learning". *IEEE International Conference on Data Mining Workshops*.

Baker, Bowen et al. (2017). "Designing Neural Network Architectures using Reinforcement Learning". *International Conference on Learning Representations*.

Ballard, Dana H. (1987). "Modular Learning in Neural Networks". *National Conference on Artificial Intelligence*.

Bar, Y. et al. (2015). "Chest pathology detection using deep learning with non-medical training". *International Symposium on Biomedical Imaging*.

Ben-David, Shai et al. (2010). "A theory of learning from different domains". *Machine Learning*.

Bengio, Yoshua et al. (2007). "Greedy Layer-Wise Training of Deep Networks". *Advances in Neural Information Processing Systems*.

Bergstra, James, Rémi Bardenet, et al. (2011). "Algorithms for Hyper-parameter Optimization". *International Conference on Neural Information Processing Systems*.

Bergstra, James and Yoshua Bengio (2012). "Random Search for Hyper-parameter Optimization". *Journal of Machine Learning Research*.

Bergstra, James, Daniel Yamins, and David D. Cox (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". *International Conference on Machine Learning*.

Braun, Heinrich and Joachim Weisbrod (1993). "Evolving Neural Feedforward Networks". *Artificial Neural Nets and Genetic Algorithms*.

Buda, Mateusz, Atsuto Maki, and Maciej A. Mazurowski (2018). "A systematic study of the class imbalance problem in convolutional neural networks". *Neural Networks*.

Cabezas, Mariano et al. (2011). "A review of atlas-based segmentation for magnetic resonance brain images". *Computer Methods and Programs in Biomedicine*.

Caruana, Rich (1995). "Learning Many Related Tasks at the Same Time with Backpropagation". *Advances in Neural Information Processing Systems*.

— **(1997)**. "Multitask Learning". *Machine Learning - Special issue on inductive transfer*.

Cerrolaza, Juan J. et al. (2014). "Segmentation of kidney in 3D-ultrasound images using Gabor-based appearance models". *IEEE International Symposium on Biomedical Imaging*.

Chang, Hang et al. (2018). "Unsupervised Transfer Learning via Multi-Scale Convolutional Sparse Coding for Biomedical Applications". *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Cheng, Xi, Li Zhang, and Yefeng Zheng (2015). "Deep similarity learning for multimodal medical images". *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*.

Chollet, François et al. (2015). *Keras*.

Çiçek, Özgün et al. (2016). "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation". *Medical Image Computing and Computer-Assisted Intervention*.

Collobert, Ronan and Jason Weston (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". *International Conference on Machine Learning*.

Commowick, Olivier and Grégoire Malandain (2007). "Efficient Selection of the Most Similar Image in a Database for Critical Structures Segmentation". *Medical Image Computing and Computer-Assisted Intervention*.

Cootes, T. F., G. J. Edwards, and C. J. Taylor (1998). "Active appearance models". *European Conference on Computer Vision*. Ed. by Hans Burkhardt and Bernd Neumann.

Cootes, T.F. et al. (1995). "Active Shape Models-Their Training and Application". *Computer Vision and Image Understanding*.

Crammer, Koby, Michael Kearns, and Jennifer Wortman (2008). "Learning from Multiple Sources". *Journal of Machine Learning Research*.

Dai, Wenyuan et al. (2007). "Boosting for Transfer Learning". *International Conference on Machine Learning*.

— **(2008)**. "Self-taught Clustering". *International Conference on Machine Learning*.

Dalca, Adrian V. et al. (2018). "Unsupervised Learning for Fast Probabilistic Diffeomorphic Registration". *Medical Image Computing and Computer Assisted Intervention*.

Daume III, Hal (2007). "Frustratingly Easy Domain Adaptation". *Annual Meeting of the Association of Computational Linguistics*.

Domhan, Tobias, Jost Tobias Springenberg, and Frank Hutter (2015). "Speeding Up Automatic Hyperparameter Optimization of Deep Neural Net-

works by Extrapolation of Learning Curves". *International Conference on Artificial Intelligence.*

Donahue, Jeff et al. (2014). "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition". *Proceedings of Machine Learning Research.*

Erhan, Dumitru et al. (2010). "Why Does Unsupervised Pre-training Help Deep Learning?" *Journal of Machine Learning Research.*

Evgeniou, Theodoros and Massimiliano Pontil (2004). "Regularized Multi–task Learning". *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Falkner, Stefan, Aaron Klein, and Frank Hutter (2018). "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". *International Conference on Machine Learning.*

Fan, Jingfan et al. (2018). "Adversarial Similarity Network for Evaluating Image Alignment in Deep Learning Based Registration". *Medical Image Computing and Computer Assisted Intervention.*

Fredrikson, Matt, Somesh Jha, and Thomas Ristenpart (2015). "Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures". *ACM SIGSAC Conference on Computer and Communications Security.*

Gao, Jing et al. (2008). "Knowledge Transfer via Multiple Model Local Structure Mapping". *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Garbuno-Inigo, A., F.A. DiazDelaO, and K.M. Zuev (2016). "Gaussian process hyper-parameter estimation using Parallel Asymptotically Independent Markov Sampling". *Computational Statistics and Data Analysis.*

Ginneken, B. van et al. (2015). "Off-the-shelf convolutional neural network features for pulmonary nodule detection in computed tomography scans". *IEEE 12th International Symposium on Biomedical Imaging.*

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning.*

Goodfellow, Ian, Jean Pouget-Abadie, et al. (2014). "Generative Adversarial Nets". *Advances in Neural Information Processing Systems.*

Gupta, Ashish, Murat Seçkin Ayhan, and Anthony S. Maida (2013). "Natural Image Bases to Represent Neuroimaging Data". *International Conference on Machine Learning*.

Hazan, Elad, Adam Klivans, and Yang Yuan (2018). "Hyperparameter optimization: a spectral approach". *International Conference on Learning Representations*.

Heckemann, Rolf A. et al. (2006). "Automatic anatomical brain MRI segmentation combining label propagation and decision fusion". *NeuroImage*.

Heimann, Tobias and Hans-Peter Meinzer (2009). "Statistical shape models for 3D medical image segmentation: A review". *Medical Image Analysis*.

Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh (2006). "A Fast Learning Algorithm for Deep Belief Nets". *Neural Computation*.

Hofmanninger, J. and G. Langs (2015). "Mapping visual features to semantic profiles for retrieval in medical imaging". *IEEE Conference on Computer Vision and Pattern Recognition*.

Jaderberg, Max et al. (2015). "Spatial Transformer Networks". *Advances in Neural Information Processing Systems*.

Jones, Donald R. (2001). "A Taxonomy of Global Optimization Methods Based on Response Surfaces". *Journal of Global Optimization*.

Joshi, Mahesh et al. (2013). "What's in a Domain? Multi-Domain Learning for Multi-Attribute Data". *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Joshi, Sarang et al. (2004). "Unbiased diffeomorphic atlas construction for computational anatomy". *NeuroImage*.

Kamnitsas, Konstantinos, Christian Baumgartner, et al. (2017). "Unsupervised Domain Adaptation in Brain Lesion Segmentation with Adversarial Networks". *Information Processing in Medical Imaging*.

Kamnitsas, Konstantinos, Christian Ledig, et al. (2017). "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation". *Medical Image Analysis*.

Krizhevsky, Alex (2009). *Learning Multiple Layers of Features from Tiny Images*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". *Neural Information Processing Systems*.

Kushner, H. J. (1964). "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise". *Journal of Basic Engineering*.

Lawrence, Neil D. and John C. Platt (2004). "Learning to Learn with the Informative Vector Machine". *International Conference on Machine Learning*.

Lee, Su-In et al. (2007). "Learning a Meta-level Prior for Feature Relevance from Multiple Related Tasks". *International Conference on Machine Learning*.

Li, Lisha et al. (2017). "Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits". *International Conference on Learning Representations*.

Marsousi, Mahdi, Konstantinos N. Plataniotis, and Stergios Stergiopoulos (2017). "An Automated Approach for Kidney Segmentation in Three-Dimensional Ultrasound Images". *IEEE Journal of Biomedical and Health Informatics*.

Miao, S., Z. J. Wang, and R. Liao (2016). "A CNN Regression Approach for Real-Time 2D/3D Registration". *IEEE Transactions on Medical Imaging*.

Miikkulainen, Risto et al. (2017). "Evolving Deep Neural Networks". *arXiv*.

Miller, Geoffrey F., Peter M. Todd, and Shailesh U. Hegde (1989). "Designing Neural Networks Using Genetic Algorithms". *International Conference on Genetic Algorithms*.

Mory, Benoît (2011). "Interactive Segmentation of 3D Medical Images with Implicit Surfaces". PhD thesis. Lausanne: IEL.

Mory, Benoît et al. (2012). "Real-Time 3D Image Segmentation by User-Constrained Template Deformation". *Medical Image Computing and Computer-Assisted Intervention*.

Murray, Iain and Ryan P. Adams (2010). "Slice Sampling Covariance Hyperparameters of Latent Gaussian Models". Neural Information Processing Systems.

Nam, Hyeonseob and Bohyung Han (2016). "Learning Multi-Domain Convolutional Neural Networks for Visual Tracking". *IEEE Conference on Computer Vision and Pattern Recognition*.

Neal, Radford M. (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics.

Oh, C., E. Gavves, and M. Welling (2018). "BOCK : Bayesian Optimization with Cylindrical Kernels". *arXiv*.

Oquab, M. et al. (2014). "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks". *IEEE Conference on Computer Vision and Pattern Recognition*.

Paciorek, Christopher J. and Mark J. Schervish (2003). "Nonstationary Covariance Functions for Gaussian Process Regression". *International Conference on Neural Information Processing Systems*.

Pan, Sinno Jialin and Qiang Yang (2010). "A Survey on Transfer Learning". *IEEE Transactions on Knowledge and Data Engineering*.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research*.

Pham, Hieu et al. (2018). "Efficient Neural Architecture Search via Parameters Sharing". *International Conference on Machine Learning*.

Prevost, Raphaël (2013). "Variational methods for model-based image segmentation - applications in medical imaging". PhD thesis.

Raina, Rajat et al. (2007). "Self-taught Learning: Transfer Learning from Unlabeled Data". *International Conference on Machine Learning*.

Rasmussen, C. E. and C. K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*.

Ravishankar, H. et al. (2017). "Learning and Incorporating Shape Models for Semantic Segmentation". *Medical Image Computing and Computer-Assisted Intervention*.

Razavian, Ali Sharif et al. (2014). "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". *IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

Real, Esteban et al. (2017). "Large-Scale Evolution of Image Classifiers". *International Conference on Machine Learning.*

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". *Medical Image Computing and Computer-Assisted Intervention.*

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". *International Journal of Computer Vision.*

Saddi, K. A. et al. (2007). "Region-Based Segmentation via Non-Rigid Template Matching". *IEEE International Conference on Computer Vision.*

Schlegl, Thomas, Joachim Ofner, and Georg Langs (2014). "Unsupervised Pre-training Across Image Domains Improves Lung Tissue Classification". *Medical Computer Vision: Algorithms for Big Data.*

Schonlau, Matthias, William J. Welch, and Donald R. Jones (1998). "Global versus Local Search in Constrained Optimization of Computer Models". *Institute of Mathematical Statistics Lecture Notes - Monograph Series.*

Sermanet, Pierre et al. (2014). "Overfeat: Integrated recognition, localization and detection using convolutional networks". *International Conference on Learning Representations.*

Shahriari, Bobak et al. (2016). "Taking the Human Out of the Loop: A Review of Bayesian Optimization". *Proceedings of the IEEE.*

Shin, Hoo-chang et al. (2016). "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning". *IEEE Transactions on Medical Imaging.*

Simonovsky, Martin et al. (2016). "A Deep Metric for Multimodal Registration". *Medical Image Computing and Computer-Assisted Intervention.*

Simonyan, Karen and Andrew Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition".

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". *International Conference on Neural Information Processing Systems.*

Snoek, Jasper, Kevin Swersky, et al. (2014). "Input Warping for Bayesian Optimization of Non-stationary Functions". *International Conference on Machine Learning*.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research*.

Stanley, Kenneth O., David B. D'Ambrosio, and Jason Gauci (2009). "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks". *Artificial Life*.

Stanley, Kenneth O. and Risto Miikkulainen (2002). "Evolving Neural Networks Through Augmenting Topologies". *Evolutionary Computation*.

Swersky, K. et al. (2013). "Raiders of the Lost Architecture : Kernels for Bayesian Optimization in Conditional Parameter Spaces".

Titsias, Michalis K., Magnus Rattray, and Neil D. Lawrence (2011). "Markov chain Monte Carlo algorithms for Gaussian processes". *Bayesian Time Series Models*.

Tompson, J. et al. (2015). "Efficient object localization using Convolutional Networks". *IEEE Conference on Computer Vision and Pattern Recognition*.

Tzeng, Eric et al. (2015). "Simultaneous Deep Transfer Across Domains and Tasks". *IEEE International Conference on Computer Vision*.

Vincent, Pascal et al. (2008). "Extracting and Composing Robust Features with Denoising Autoencoders". *International Conference on Machine Learning*.

Warfield, S. K., K. H. Zou, and W. M. Wells (2004). "Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation". *IEEE Transactions on Medical Imaging*.

Wu, Guorong et al. (2013). "Unsupervised Deep Feature Learning for Deformable Registration of MR Brain Images". *Medical Image Computing and Computer-Assisted Intervention*.

Yang, Xiao, Roland Kwitt, and Marc Niethammer (2016). "Fast Predictive Image Registration". *Deep Learning and Data Labeling for Medical Applications*.

Yezzi, Anthony J. and Stefano Soatto (2003). "Deformotion: Deforming Motion, Shape Average and the Joint Registration and Approximation of Structures in Images". *International Journal of Computer Vision*.

Yosinski, Jason et al. (2014). "How Transferable Are Features in Deep Neural Networks?" *International Conference on Neural Information Processing Systems*.

Zela, Arber et al. (2018). "Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search". *arXiv*.

Zoph, B. and Q. V. Le (2017). "Neural Architecture Search with Reinforcement Learning". *International Conference on Learning Representations*.

Zoph, Barret et al. (2017). "Learning Transferable Architectures for Scalable Image Recognition". *arXiv*.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Optimisation d'hyper-paramètres en apprentissage profond et apprentissage par transfert - Applications en imagerie médicale

Mots clés : Apprentissage profond, imagerie médicale, apprentissage par transfert, déformation de modèles, segmentation

Résumé : Ces dernières années, l'apprentissage profond a complètement changé le domaine de vision par ordinateur. Plus rapide, donnant de meilleurs résultats, et nécessitant une expertise moindre pour être utilisé que les méthodes classiques de vision par ordinateur, l'apprentissage profond est devenu omniprésent dans tous les problèmes d'imagerie, y compris l'imagerie médicale.
Au début de cette thèse, la construction de réseaux de neurones adaptés à des tâches spécifiques ne bénéficiait pas encore de suffisamment d'outils ni d'une compréhension approfondie. Afin de trouver automatiquement des réseaux de neurones adaptés à des tâches spécifiques, nous avons ainsi apporté des contributions à l'optimisation d'hyper-paramètres de réseaux de neurones. Cette thèse propose une comparaison de certaines méthodes d'optimisation, une amélioration en performance d'une de ces méthodes, l'optimisation bayésienne, et une nouvelle méthode d'optimisation d'hyper-paramètres basé sur la combinaison de deux méthodes existantes : l'optimisation bayésienne et hyperband.
Une fois équipés de ces outils, nous les avons utilisés pour des problèmes d'imagerie médicale : la classification de champs de vue en IRM, et la segmentation du rein en échographie 3D pour deux groupes de patients. Cette dernière tâche a nécessité le développement d'une nouvelle méthode d'apprentissage par transfert reposant sur la modification du réseau de neurones source par l'ajout de nouvelles couches de transformations géométrique et d'intensité.
En dernière partie, cette thèse revient vers les méthodes classiques de vision par ordinateur, et nous proposons un nouvel algorithme de segmentation qui combine les méthodes de déformations de modèles et l'apprentissage profond. Nous montrons comment utiliser un réseau de neurones pour prédire des transformations globales et locales sans accès aux vérités-terrains de ces transformations. Cette méthode est validé sur la tâche de la segmentation du rein en échographie 3D.

Title : Hyper-parameter optimization in deep learning and transfer learning - Applications to medical imaging

Keywords : Deep learning, medical imaging, transfer learning, template deformation, segmentation

Abstract : In the last few years, deep learning has changed irrevocably the field of computer vision. Faster, giving better results, and requiring a lower degree of expertise to use than traditional computer vision methods, deep learning has become ubiquitous in every imaging application. This includes medical imaging applications.
At the beginning of this thesis, there was still a strong lack of tools and understanding of how to build efficient neural networks for specific tasks. Thus this thesis first focused on the topic of hyper-parameter optimization for deep neural networks, i.e. methods for automatically finding efficient neural networks on specific tasks. The thesis includes a comparison of different methods, a performance improvement of one of these methods, Bayesian optimization, and the proposal of a new method of hyper-parameter optimization by combining two existing methods: Bayesian optimization and Hyperband.
From there, we used these methods for medical imaging applications such as the classification of field-of-view in MRI, and the segmentation of the kidney in 3D ultrasound images across two populations of patients. This last task required the development of a new transfer learning method based on the modification of the source network by adding new geometric and intensity transformation layers.
Finally this thesis loops back to older computer vision methods, and we propose a new segmentation algorithm combining template deformation and deep learning. We show how to use a neural network to predict global and local transformations without requiring the ground-truth of these transformations. The method is validated on the task of kidney segmentation in 3D US images.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France