Making the most of available data : representation and adaptation for few-shot image classification

Yann Lifchitz

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ RENNES 1

ÉCOLE DOCTORALE Nº 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Signal, Image, Vision*

Par

Yann LIFCHITZ

Making the most of available data: representation and adaptation for few-shot image classification

Thèse présentée et soutenue à Rennes, le 20 Avril 2021
Unité de recherche : Inria Rennes Bretagne

Rapporteurs avant soutenance :

Nikos KOMODAKIS	Professeur, Université d'Heraklion Crète
Stéphane CANU	Professeur, INSA Rouen Normandie

Composition du Jury :

Présidente :	Elisa FROMONT	Professeur, Université de Rennes 1
Examinateurs :	Frederic JURIE	Professeur, Université de Caen Normandie
	Patrick PEREZ	Directeur scientifique, Valeo.ai Paris
	Nikos KOMODAKIS	Professeur, Université d'Heraklion Crète
	Stéphane CANU	Professeur, INSA Rouen Normandie
Dir. de thèse :	Yannis AVRITHIS	Chargé de recherche, Inria Rennes Bretagne
Encadr. de thèse :	Sylvaine PICARD	Responsable industriel, Safran

Invité :

Andrei BURSUC	Chercheur, Valeo.ai Paris

Acknowledgements

I would like to thank all of the people that contributed to this thesis. As part of the Cifre process, I had the chance to work with many people during three years.

I was very fortunate to have Yannis Avrithis as my director, his incredible dedication to research and helping his students was a considerable help throughout the thesis. On the Safran side, I want to thank Sylvaine Picard for her trust, guidance and ideas that were always very useful. I also want to thank Andrei Bursuc, who also advised me at the beginning of this work, setting me on the right track.

Thank you to Nikos Komodakis and Stéphane Canu that reviewed this manuscript and gave valuable feedback as well as Elisa Fromont, Patrick Perez and Frédéric Jurie for accepting to be part of my jury.

While at Inria and Safran, I was surrounded by many wonderful colleagues: thank you all for making those years more enjoyable. Finally I would thank my friends and family that were always there to support me.

Contents

List of Figures

List of Tables

Résumé en français

Ces dernières années ont connu l'émergence de nouvelles techniques de vision par ordinateur, permettant d'énormes progrès dans les tâches populaires telles que la classification d'image, la détection d'objets ou la segmentation sémantique. Dans certains cas, les performances atteintes dépassent même celles des humains. C'est notamment les cas de la classification d'image, comme démontré par les résultats de la compétition annuelle ImageNet. L'objectif du challenge est de développer des modèles capables de classifier automatiquement des images dans 1000 catégories. Depuis 2015, les modèles vainqueurs ont des taux d'erreurs inférieurs à celui d'un humain moyen. On peut expliquer ce succès par la concordance de l'apparition de nouveaux modèles, l'accessibilité rapide à des vastes bases de données ainsi que le développement d'unités de calculs. Plus spécifiquement, une partie du monde de la recherche en vision par ordinateur s'est focalisé sur la conception et l'utilisation de réseaux de neurones profonds. Ces modèles contiennent des millions de paramètres, qui, après avoir été réglés, peuvent approximer des fonctions permettant de réaliser des tâches complexes, telles que le calcul de la probabilité qu'une image appartiennent à une catégorie. Le réglage des paramètres est un problème d'apprentissage statistique. Des exemples sont montrés aux modèles, les paramètres sont ajustés grâce à un algorithme d'optimisation pour apprendre à bien réaliser la tâche sur ceux-ci. L'apprentissage de l'ensemble des paramètres nécessite un grand nombre d'exemples et est coûteux en ressources. Les GPUs modernes permettent la parallélisation des opérations sous-jacentes, rendant l'utilisation de réseaux profonds accessible.

Modèles pour la classification

Une tâche commune à toute application de vision par ordinateur utilisant des images comme entrées est l'obtention d'une bonne méthode de représentation de celles-ci. Les images sont généralement encodées numériquement sous la forme de séquences de valeurs

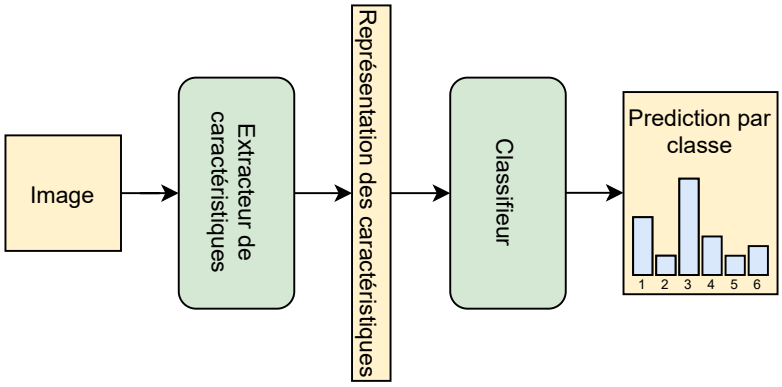


Figure 1 – Structure générique d'un modèle pour la classification d'images. L'extracteur de caractéristiques renvoie une description de l'image facilitant la classification. Le classifieur utilise la représentation pour produire une probabilité d'appartenance pour chaque catégorie.

discrètes correspondant à l'intensité de chaque pixel. Il est difficile d'utiliser directement cette représentation pour effectuer des tâches complexes telles que la classification. Une représentation plus haut niveau, non pas selon les pixels mais selon le contenu de l'image, est nécessaire, de la même manière qu'un humain peut juger de la nature des objets en identifiant certains attributs remarquables.

Par conséquent, les modèles de vision par ordinateurs peuvent contenir deux étapes. Tout d'abord, un modèle d'extraction de caractéristiques transforme la représentation sous forme de pixels en représentation de caractéristiques. Les représentations de caractéristiques n'ont pas pour but de décrire parfaitement les images, à la place ils véhiculent des informations sémantiques sur ce qui se trouve dans les images.

De nombreuses fonctions de représentation d'images ont été proposées. Certaines sont définies par un algorithme fixe, ces algorithmes découlant de la compréhension des chercheurs sur la structure des images. D'autres sont basées sur les méthodes d'apprentissage automatique, auquel cas la fonction est apprise sur un ensemble d'exemples d'images. Une revue récente ces méthodes est [Ma+21].

Une fois l'image convertie en une représentation haut niveau, un autre modèle la traite pour effectuer la tâche souhaitée. Dans le cas de la classification d'images, le classifieur produit un score ou une probabilité d'appartenance pour chacune des classes possibles. Ce fonctionnement générique est illustré dans Figure 1

Extraction de caractéristiques par apprentissage profond

Les fonctions d'extraction de caractéristiques fixes sont des fonctions génériques crées pour pouvoir s'appliquer sur différents type d'images. Bien que ces représentations soient suffisantes pour certaines tâches, idéalement, la fonction de représentation devrait être adaptée à la tâche à résoudre. C'est ce que permettent les méthodes basées sur l'apprentissage automatique. En particulier, les réseaux de neurones convolutifs se sont récemment avérés produire de puissantes représentations des images. En plus de remplacer les extracteurs de caractéristiques, les réseaux de neurones peuvent également remplir le rôle du classifieur, unifiant les deux parties de l'approche de la Figure 1

Les réseaux de neurones sont constitués d'unités de calcul paramétrables appelées neurones. Ceux-ci sont organisés en couches successives, de sorte que l'entrée d'un neurone est la sortie des neurones de la couche précédente. Dans certains réseaux dits convolutifs, certaines connexions entre neurones sont désactivées, permettant ainsi de réaliser des opérations de convolution permettant de mieux traiter les images. Les convolutions extraient des informations localisées dans l'image. En conséquence, la première couche d'un réseau neuronal peut apprendre à extraire des informations de bas niveau localement, par exemple le type de texture. Les couches suivantes combinent ces informations pour l'extraction d'informations de plus haut niveau et moins localisées. Les architectures récentes de réseaux de neurones sont dites profondes, c'est-à-dire qu'ils sont constitués de nombreuses couches, plus de 100 dans les cas extrêmes. Ils contiennent des millions de paramètres qui doivent être régler à l'aide de méthodes d'optimisation.

Lorsque des données annotées, c'est-à-dire dont on connaît la catégorie, sont accessible, le réglage des paramètres peut passer par l'*apprentissage supervisé*. Lors de cet apprentissage les données sont traitées par le réseau qui fournit les prédictions concernant les catégories. Une fonction de perte évalue l'écart entre les prédictions avec la vérité. Des algorithmes d'optimisation sont utilisés pour minimiser la fonction de perte en ajustant les paramètres.

Classification few-shot

Les modèles basés sur les réseaux de neurones profonds nécessitent une énorme quantité de données annotées pour la phase d'apprentissage pour atteindre des performances rivalisant avec l'humain. Par exemple, la base de données ImageNet [Rus+14] contient des millions d'images annotées. Le grand nombre d'images d'entraînement est important pour que les bonnes performances pendant la phase d'apprentissage se généralisent

à des nouvelles données, jamais vues par le modèle. En pratique, dans de nombreux cas, une telle quantité de données n'est pas accessible. Par conséquent, il est important de trouver des méthodes réduisant cette dépendance à la supervision. Cela a motivé les travaux sur l'*apprentissage non-supervisé*, où les données sont accessibles mais non annotées, l'*apprentissage semi-supervisé*, où seul une partie des données sont annotées, l'*apprentissage actif* où le modèle choisit un sous-ensemble d'images à faire annoter on encore l'*apprentissage par transfert*, où la connaissance accumulées par l'apprentissage d'une autre tâche est adaptée à la tâche en cours. Dans tous ces domaines, le nombre d'exemples disponibles par l'apprentissage reste relativement grand, de l'ordre des milliers.

Pousser à l'extrême la contrainte sur les données d'apprentissage amène à définir l'*apprentissage frugal*, ou *few-shot learning*, pour lequel les données d'entraînement sont annotées mais limitées à quelques exemples, appelés *exemples support*, des dizaines au maximum. Dans le cas extrême du one-shot learning, un seul exemple par catégorie est utilisable. Avec si peu de données, le problème de généralisation entre les données d'entraînement et de test est plus difficile. Les exemples d'apprentissage ne représentent que des définitions incomplètes de leurs catégories, ne décrivant pas les variabilités d'apparences au sein des classes. Il est donc possible que certaines images à classifier ne ressemblent à aucunes de celles vues lors de l'apprentissage. Avec aussi peu de supervision, l'entraînement complet d'un réseau de neurones profond n'est pas réalisable.

En comparaison, les humains possèdent la capacité d'être capable d'apprendre à reconnaître de nouveaux objets, et plus généralement, à effectuer des nouvelles tâches en se basant sur un nombre limité d'observations. Cela peut s'expliquer par l'accumulation de connaissances préalables au cours de la vie, permettant une meilleure compréhension de la structure des scènes ainsi que des méthodes générales de résolution de tâches. De la même façon, les méthodes de few-shot learning, utilisent de connaissances extérieures à la tâche, sous la forme d'une base de données plus grande avec des catégories disjointes, pour apprendre à bien représenter les images pour la classification. Cette base de données est appelée *données de base*. Certaines méthodes vont plus loin, en faisant du meta-apprentissage, ce qui consiste à apprendre des méthodes permettant de mieux apprendre à résoudre tout type de tâches. Ces méthodes utilisent alors la base de connaissance à priori pour générer artificiellement des tâches few-shot afin d'apprendre à les résoudre.

Après cette étape préliminaire que nous appellerons étape d'apprentissage de représentation, les quelques exemples supports de la tâche few-shot sont utilisés, en combinaison avec la fonction de représentation obtenue, pour former un modèle de classification dans les nouvelles catégories. Ce processus est similaire à celui mis en place dans le cas de

l'apprentissage par transfert. A la différence que pour l'apprentissage par transfert, la fonction de représentation des images est généralement adaptée aux nouvelles données lors d'une nouvelle phase d'entraînement sur les paramètres du réseau. Cette étape d'adaptation est plus difficile à réaliser dans le cas du few-shot. Plus le modèle utilisé contient de paramètres, plus il faut de données pour les ajuster, sinon le modèle sur-apprend, c'est-à-dire est efficace sur les données d'apprentissage mais ne généralise plus aux autre données. Par conséquent, l'adaptation est généralement limitée à la construction d'un classifieur pour les nouvelles classes, qui agit sur la sortie de la fonction de représentation. Les classifieurs sont des modèles d'apprentissage automatique avec peu de paramètres ou des modèles non-paramétriques pour éviter le surapprentissage.

Solutions

Dans cette thèse, nous abordons le problème de la classification frugale en se concentrant sur l'apprentissage d'une bonne fonction de représentation des images et en utilisant les quelques exemples supports pour l'adapter au mieux pour la tâche. En outre, dans le souci d'utiliser au mieux toutes les données accessibles, nous étendons le cadre de la classification few-shot standard à travers des tâches plus réalistes. Le diagramme de la Figure 2 résume la structure utilisée pour classifier une image dans une tâche few-shot. Nos contributions sont notées en bleu.

Apprentissage de représentation

L'apprentissage de représentation se fait grâce à l'utilisation des données de base. Après avoir défini des méthodes pour la validation des paramètres pour la phase d'apprentissage de représentation, on compare les efficacités de deux méthodes répandues : prototypical networks et le classifieur cosinus. Les prototypicals networks apprennent à résoudre une multitude de tâches échantillonnées dans les données de base. Tandis que le modèle du classifieur cosinus est plus simple, il s'agit d'entraîner un réseau à résoudre un problème de classification sur toutes les classes de base, puis de récupérer une partie du réseau qui est interprétée comme fonction d'extraction de caractéristiques. On montre que le classifieur cosinus, malgré sa simplicité est la méthode la plus prometteuse lorsque l'on utilise des architectures de réseaux profonds telles que les residual networks. Nous proposons la classification dense comme amélioration de cette méthode. En sortie du réseau d'extraction de caractéristiques, les représentations sont des tenseurs 3d qui sont généralement ramenés à des vecteurs en réduisant les dimensions spatiales. Avec la

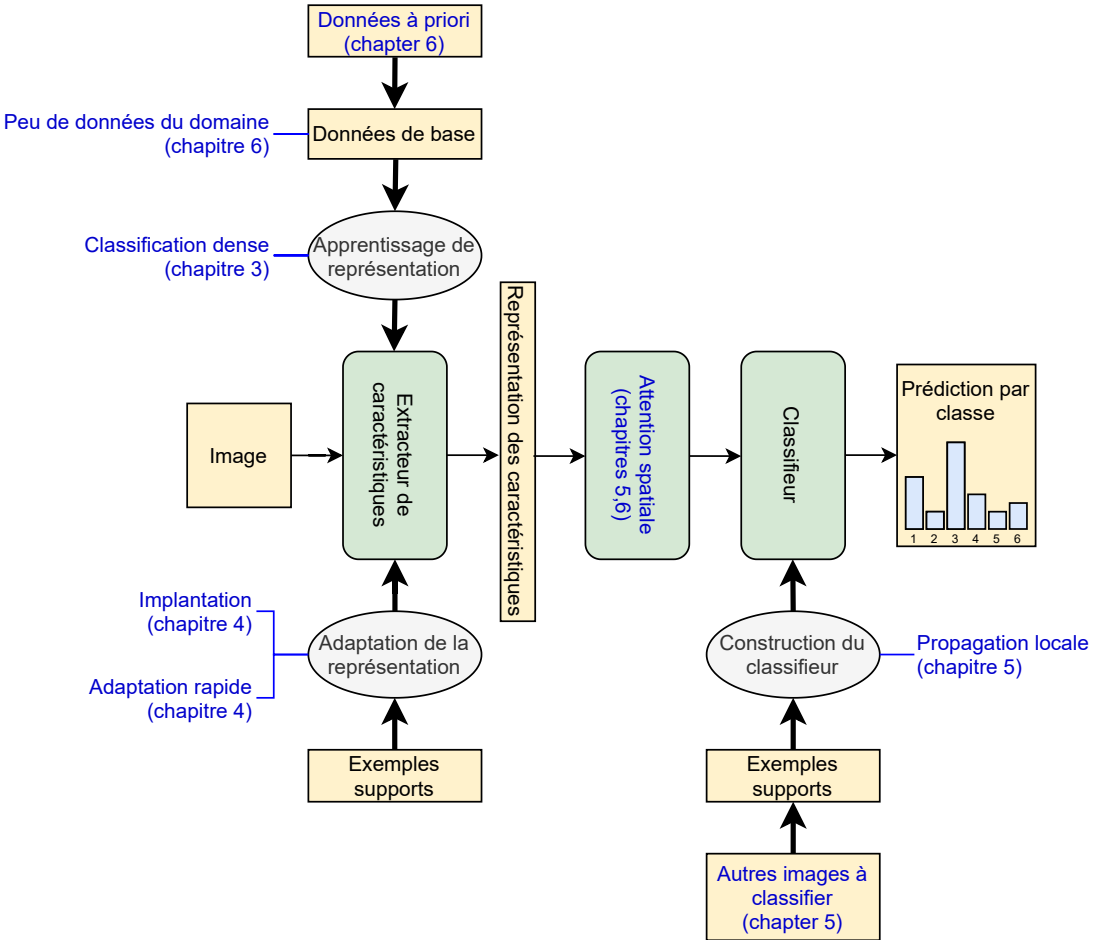


Figure 2 – Structure proposée pour la classification d'un exemple ans un problème de classification few-shot. La fonction d'extraction de caractéristique est entraînée pendant la phase d'apprentissage de la représentation utilisant les données de base puis adaptée avec les quelques exemples. Un classifieur pour les classes few-shot est construit en utilisant les quelques exemples. Une opération d'attention spatiale est utilisée pour sélectionner les représentations locales pertinentes. En bleu sont notées nos contributions et les chapitres dans lesquelles elles sont développées.

classification dense, on propose de ne pas effectuer cette réduction de dimensions, et d'utiliser chaque position spatiale comme une représentation locale de l'image. Au lieu de réunir ces représentations, on entraîne le réseau à classifier chacune des représentations locales dans la classe de l'image. Ainsi, on apprend alors une fonction de représentation plus riche qui permet d'améliorer les performances de classification, notamment sur les nouvelles tâches few-shot.

Adaptation de la fonction de représentation

La fonction de représentation apprise sur les données de base n'est pas adaptée aux tâches few-shot que l'on veut résoudre, étant donné que les données de base appartiennent à des classes disjointes des classes few-shot. Dans cette thèse nous allons nous intéresser aux méthodes permettant d'effectuer cette adaptation tout en évitant le problème du surapprentissage. Nous proposons deux solutions. Tout d'abord l'implantation, méthode qui consiste à ajouter en parallèle du réseau des nouvelles couches de convolution appelés implants. Les paramètres des implants sont appris en sur les exemples supports seulement, ils sont donc spécifiques à la tâche few-shot considérée. Ils permettent de calculer des caractéristiques spécifiques à la tâche, qui sont concaténées aux caractéristiques calculées par le réseau entraîné auparavant. Comme les paramètres à apprendre sont moindre, on peut contrôler le surapprentissage, permettant alors d'apprendre sur les exemples supports et d'améliorer la classification de nouveaux exemples. La deuxième méthode consiste à ajuster plusieurs couches du réseau d'extraction de caractéristiques avec un petit taux d'apprentissage et pour un petit nombre d'itérations seulement. Le nombre d'itération d'apprentissage à effectuer est déterminé par observation du comportement moyen des courbes d'apprentissage sur une base de données de validation. Bien que très simple, cette approche permet une amélioration notable des performances dans de nombreux cas. Dans le cas extrême du one-shot learning, cette méthode est peu efficace car le surapprentissage est quasi-immédiat. Dans ce cas, on montre que l'on peut augmenter le nombre d'exemples support artificiellement en réutilisant des images des classes de bases proches des exemples supports et en les assimilant à la nouvelle tâche.

Propagation locale

On propose également de s'intéresser au problème de la classification transductive, dans lequel plusieurs images doivent être classifiées en même temps. Les images à classifier peuvent être utilisées comme exemples non-annotés. La prise en compte de ces exemples est d'autant plus importante dans le cadre du few-shot learning ou les données sont rares.

Une façon d'en tirer parti est de construire un graphe dans lequel les sommets sont les exemples (annotées et non-annotées), puis de propager l'information de classe dans le graphe pour classifier tous les sommets. Nous proposons une nouvelle version de cette méthode consistant à augmenter artificiellement le nombre d'exemples en considérant un sommet par représentation locale. La classification se base alors sur des comparaisons locales entres les images. Dans ce contexte on montre l'importance de sélectionner uniquement les régions pertinentes des images. On introduit alors un mécanisme simple d'attention spatiale qui filtre les informations liées au fond des images. Ce mécanisme est important pour la propagation locale mais on montre qu'elle apporte également des améliorations de performances lorsque combinée avec d'autres méthodes.

Few-shot Few-shot learning

Dans les problèmes de few-shot standard, on présume l'accès aux données de base du même domaine que les données few-shot (contenu similaire), et que celles-ci soient assez nombreuses pour l'apprentissage d'une fonction de représentation. Cela n'est pas toujours le cas, d'autant plus que le few-shot s'applique dans des domaines pour lesquels l'information est difficilement accessible. Par contre, sont ignorées la multitude de grandes bases de données accessibles publiquement. Ces données ne sont pas nécessairement dans le même domaine mais on peut se demander si elles peuvent être utilisées pour les tâches few-shot. Nous proposons un nouveau cadre few-shot dans lesquels les données de bases sont peu nombreuses et que l'on peut utiliser des données extérieures. En l'occurrence, nous utilisons un réseau pré-entraîné sur la base Places. Si les données de base sont accessibles, alors on peut adapter le réseau au domaine. Puis, le réseau peut être une nouvelle fois adapté avec les exemples support. Dans ce contexte, on introduit un autre nouveau mécanisme d'attention spatiale basé sur le classifieur pré-entraîné, permettant de focaliser l'attention sur les régions pertinentes de l'image. On montre que dans tous les cas, l'utilisation d'un réseau pré-entraîné est bénéfique. De plus, on montre que quelques exemples de bases suffisent pour augmenter grandement les performances.

Application aux images aériennes

Nous avons également testé une partie de nos méthodes sur une application spécifique, la classification de véhicules rares dans des images aériennes. Cette tâche comprenant des spécificités propres, telles que le faible nombre de classes ainsi que la présence d'une classe de fonds, qu'une étude spécifique future pourra étudier. On montre néanmoins que la classification dense semble être bénéfique. De plus, les mécanismes d'attention

semblent pouvoir se généraliser.

Chapter 1

Introduction

Contents

In the last few years, novel computer vision algorithms have made huge progress on popular tasks such as image classification, object recognition or semantic segmentation, exceeding sometimes human visual abilities. The results of the annual ImageNet challenge are a testament of those improvements. The objective of this challenge is to develop models able to classify images into 1000 classes. Since 2015, the winning model of this challenge has error rate smaller than that of the average human. Such successes are due to a concordance of the use of novel large models, access to large-scale datasets, and powerful hardware to manage the two. More specifically, computer vision research has refocused on the use of deep neural networks. Those models contain millions of parameters, which, once tuned, can approximate a complex function capturing a high-level concept such as an image belonging to a class. Tuning of the parameters is a machine learning problem. A model is shown input examples and is trained by adjusting its

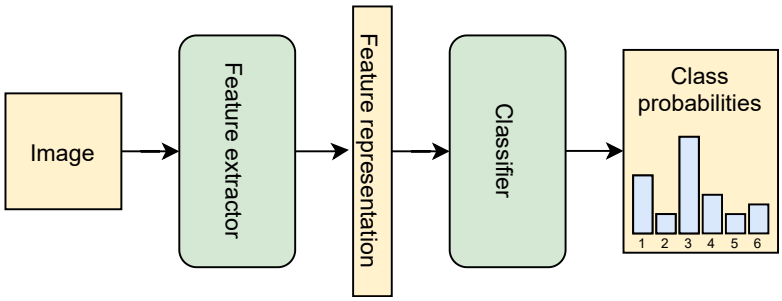


Figure 1.1 – Standard structure for an image classification model. The feature extractor extract relevant description of the image, which is then treated by the classifier. For each class, the classifier outputs the probability of the image being from the class.

parameters to perform better on those. To tune all parameters, it is necessary to use a large number of examples. The use of deep models and their training is computationally expensive. Modern GPUs allow paralellization of the underlying operations, which makes them a viable solution.

1.1 Traditional framework

One of the challenges that come with designing models that automatically interpret images is to find a good image representation. Images are digitally encoded as sequences of discrete pixel values. Pixel information is hard to use for advanced tasks such as image classification. A higher-level representation is needed for such tasks, in the same manner that humans are able to recognize objects because they can identify their attributes in the scenes.

For this reason, common computer vision models contain two stages. First, a feature extraction model transforms the pixel representation to features representation. Feature representations are not meant to perfectly describe the images at pixel level, rather carry semantic information about what is in the image they encode.

Many image representation functions have been proposed. They can either be handcrafted, that is, have a fixed definition based on expert understanding of the structure of images, or learning-based, where the function is learned on a set of image examples. A recent survey of those methods can be found in [Ma+21].

Once the image is converted to a higher-level representation, another model treats it to complete the desired task. In the case of image classification, the classifier outputs a score or probability of belonging for each of the possible classes. This general framework is illustrated in Figure 1.1.

1.1.1 Hand-crafted descriptors

Hand-crafted descriptors can either be global, in which case a unique feature vector represents an image, or local, in which case the regions of interest are each encoded separately. *Local features* can be used independently to perform region-to-region matching between two images, which can be useful for tasks such as image retrieval, stereo vision, object tracking or panorama stitching. When aggregated, local features produce powerful global representations.

Local features can be computed densely using a grid fitted on the image [TLF08], or be computed on a selection of regions of interest. In the latter case, an extra step of locating the regions of interest is introduced. Detectors aim to detect regions of the image that can be consistently detected on different views of the same scene, which are salient regions of the image. More specifically, most methods result in finding sharp edges or corners in the image. A common way of performing this detection is to track sudden variations of intensity, as done in the popular Harris corner method [HS88]. Alternatively, hybrid methods between dense and sparse local features have also been proposed [Tuy10; Isc+15].

Representations are then obtained by applying local filtering operations. That is, pixel values are combined with their neighbors by application of a filter. Filters reveals low-level image information such as frequency and scale. SIFT [Low04] and HOG [DT05] accumulate compute the direction of the change in intensity in small patches then accumulate them into histograms. GIST [OT06] uses a Gabor filter bank that act as directional edge detectors, in doing so information about the direction of edges is extracted similarly to [Low04]. Evidences show that the early visual cortex of mammals uses similar space frequency decomposition [Eve+98].

Multiple aggregation methods have been proposed to go from local features to a global representation. Bag of words is a popular method [Wil+04]. After computing local features of many different images, they are grouped together to produce a vocabulary of features. Given this vocabulary, local features of a given image are approximated by the closest visual word. The global representation is the frequency of appearance of each visual word, effectively describing what appears in the image. The VLAD approach [Jég+10] also builds a vocabulary, however the final global representation is given by the sum of the residuals between the local features and the closest visual word.

1.1.2 Classifiers

Representations of images are then input to a model that aims to achieve the desired task. In the case of image classification, the task is to predict the class of the input image. Classes being the category of the image. Class definition depends on the intended task. For instance, in an animal classification context, classes can be very broad ("animal" and "other"), individual animal ("cats", "dogs"...), or fine-grained, for instance all different subspecies. As shown in Figure 1.1, the feature representation is the input to the classifier, which usually outputs probabilities or scores for the considered classes. Such classifier must be specialized to the task. To learn a classifier, a set of image examples with the corresponding labels, called training dataset, is needed as prior knowledge of the task. The training dataset is used for learning a machine learning model. Parameters of the classifier model are tuned in accordance with the training examples.

Among the possible classifiers for image classification, *naive Bayes* [GHM07] and *support vector machines* (SVM) are linear. They require few data to train but require a good representation of the images. More specifically, image representations of different classes should be separable by a hyperplane. Non-linear classifiers such as *multi-layer neural networks* can handle intertwined representations of different classes. However, they require more examples to train. SVMs can also be made non-linear by the use of kernels; the choice of kernel requires prior knowledge of the task. Non-parametric classifiers such as the *nearest neighbor classifier* [Men+13] does not use trainable parameters. Rather, information about the examples are stored for future use. Performance is highly dependent on the quality of the representation and they are expensive as they rely on comparisons with all seen examples.

1.2 Deep learning methods

The hand-crafted features discussed above are fixed functions, designed by researchers to extract relevant information from images. The representation is not adapted to the images used for the desired tasks. Ideally, we would like a representation of the image that is optimal for the task. This is performed by replacing the hand-crafted extractors by learned representations. In particular, *convolutional neural networks* have recently proven to produce powerful representations of images. On top of being able to replace the feature extractors, neural networks can also fit the role of the classifier. The framework of Figure 1.1, consisting of separate feature extractor and classifier, is simplified to a single model performing both tasks.

Neural networks are composed of parameterizable computing units called *neurons*. Neurons are organised in *layers*. Neurons take as input the output of the previous layers, so that the data is sequentially processed layer per layer. When handling image inputs, *convolutional neural networks* (CNN) are the structure of choice. In those networks, some connections between neurons are disabled, implementing convolution operations that can extract localized information in the image. As a result, the first layer of a convolutional neural network extracts low-level localized information about the input image. Subsequent layers combine the information to extract higher-level and less localized information. Recent neural network architectures contain many layers, hence the name *deep neural networks*. The number of parameters they contain is in the order of millions. Careful tuning of those parameters must be performed using optimization algorithms.

The huge number of parameters of deep neural networks allow them to create powerful extractors. However, strong extractors can only be achieved with a strong training of the parameters. When training labeled data is available, parameters can be obtained through *supervised learning*. When training, the network learns to map an image to the corresponding output label. Parameters are learned by iteratively passing through the training data. A *loss function* determines the performance of the parameters. Optimization algorithms are used to update the parameter to minimize the loss function, resulting in maximization the performance over the whole dataset.

1.3 Few-shot classification

Deep learning models that can compete with human image classification abilities require a huge amount of labeled data. For instance, the ImageNet dataset [Rus+14] contains millions of annotated images that are used during the training stage. The large amount of training images is important for high performance on the training set after training to generalize to high performance on new data. The huge amount of data needed for training can be problematic in many practical cases where such data is not accessible. For this reason, reducing the need for supervision is becoming increasingly important. This has motivated works on *unsupervised learning*, where data is available but not annotated, *semi-supervised learning*, where only part of the data is annotated, *active learning* where the model chooses a subset of images to annotate or *transfer learning*, where knowledge gathered on another task is adapted for the current one. Still, in those cases, examples available for training are in the order of thousands. In *few-shot learning* training data is annotated but limited to few examples, tens at maximum. The

extreme case is *one-shot learning* where only one example per class is available. *Zero-shot learning* also exists where no example for the task is given, however in this case, information about the classes are given through a list of attributes, which make this problem dissimilar to few-shot learning.

With so few data, the generalization problem between the training and testing dataset is more challenging. The training data does not cover the range of appearance of the class, so we might have to classify images with few similarities to what observed at training. With so little supervision, training a deep learning model for feature extraction is not feasible.

In contrast, humans display an impressive ability to learn to recognize new objects and more generally to perform new tasks after encountering a limited number of examples. This can be explained by an accumulation of prior knowledge, allowing a better understanding of scenes description as well as a general methods for performing tasks. Similarly, few-shot learning methods use prior knowledge in accessing a larger dataset, called base dataset, disjoint from the few-shot classes to learn a good representation function. Some methods also approach the problem through *meta-learning* [FAL17; RL17; Wan+18a; NAS18], that is, learning how to solve similar problems. The latter methods use the base dataset to create artificial few-shot tasks and learn models able to solve those.

After this preliminary stage, that we will call the representation learning stage, the few examples of the novel tasks are used to build a model that performs the few-shot classification task, similarly to what is done in transfer learning. The difference is that usually transfer learning methods adapt the representation function of the image with the new data by continuing learning the network parameters. This adaptation stage is harder to perform in the few-shot case. The more parameters the machine learning model contains, the more data is needed for tuning the parameters, otherwise the model overfits, that is, it performs well on the training data but does not generalize. Therefore, adaptation is usually limited to building a classifier for the few-shot classes on top of the representation function from the representation stage. Classifiers are simple machine learning models or non-parametric classifiers similar to the one used with hand-crafted descriptors.

1.4 Objectives and contributions

Here we introduce the objectives of the thesis, namely, improving the representation of images, providing solutions to adapt the representation to a few-shot task, and rethinking

the role of data in a few-shot learning context. Our contributions related to those three objectives are briefly introduced, a more detailed description is given in chapter 2.

1.4.1 Representation

With impressive results of deep learning methods for classification, few-shot learning has recently become a very active area of research. Early works treat the problem as finding new optimization methods that are suited for learning on small data using meta-learning methods. Meta-learning is appealing for few-shot problems, because it allows building a wide variety of few-shot tasks from a large dataset, on which learning is straightforward and training for a particular few-shot task is fast, given the limited training set. An underlying hypothesis is that learning on multiple tasks results in models that are task-independent. However, since the tasks are generated from a limited base dataset, there is no guarantee that the learned model is not specialized for base classes. We argue that the most crucial element for few-shot learning is learning a good representation, which has recently been advocated in other works [Che+19; Wan+19; Gid+19; Man+20; Tia+20]. It can be achieved through meta-learning but also through simple methods that we will discuss in this thesis. In representation learning, we use the base dataset. Such dataset is said to be *in-domain*, that is, containing images from classes disjoint from the few-shot classes, but similar in content. For instance, when we learn a representation using images of some animal species, this function should generalize to other unseen animal species that correspond to the few-shot task.

In this thesis, we aim at improving the representation by studying for the first time local representation in the context of few-shot learning. In particular, during the representation learning stage, we take advantage of the CNN returning dense local features of the image to devise a training method, called *dense classification*. Dense classification encourages learning a wide variety of local details for maximum generalization to other classes. When handling examples of novel classes, we also propose spatial attention mechanisms that filters out background information, resulting in a more relevant representation of novel classes. Furthermore, using local representations of few-shot images allows to classify based on local similarities between examples, which we show to be a good classification method.

1.4.2 Adaptation

The representation function learned during the representation learning stage is based solely on examples from the base dataset. Thus, it is optimal for treating images from

this dataset. Assuming the base classes are diverse, we can hope to approach a task-independent function. For instance, in the example of animal species, if the base dataset covers all types of animals, the learned representation will probably generalize to the animal classes of the few-shot dataset. Then, a simple solution is to use the representations of few-shot examples to build a simple classifier without modifying the representation function. However, even better than a task-independent representation would be to have a task-specific function for the few-shot task. This is generally performed in transfer learning methods by adapting the representation function in a new training phase. In the case of few-shot learning, such adaptation is risky as it is prone to overfitting. For this reason, adaptation is rarely attempted in few-shot learning methods outside of meta-learning.

We propose two methods to adapt the representation function to the few-shot tasks without overfitting and without resorting to complex meta-learning methods. First, *implanting*, limits the number of task specific parameters to fine-tune with the few available data, simplifying the adaptation process. Alternatively, we show that performing standard fine-tuning can be beneficial if training is limited to a few parameter updates.

1.4.3 Role of data

In the standard few-shot learning setting, we have a fairly large set of labeled in-domain images as base dataset, then queries are treated independently at inference. We argue that this setting is often unrealistic. Having access to a base dataset of labeled in-domain examples can be difficult, especially in the case of few-shot learning where access to data is rare by definition. Moreover this setting ignores extra data that can be accessed in real life, for example, other distinct datasets may be abundant, or several queries may be available at the same time.

In this thesis, we expand on the representation learning stage by studying the special case where in-domain data is not available or limited to a few examples, introducing a new few-shot learning setup called *few-shot few-shot learning*. In this case, we propose to take advantage of a large-scale *out-of-domain* dataset, that is, with images from classes outside of the few-shot classes domain. We discuss the impact of the final classification accuracy on the few-shot task. We also also devise *local propagation*, a method that can handle classification of multiple images at once if available, leveraging other queries as unlabeled data for improved performance.

1.5 Outline

The first chapters propose solutions for the standard few-shot learning setting, with a focus on representation learning and adaptation with the few examples. In the subsequent chapters, we expand the scope beyond the standard setting to study more realistic settings.

In chapter 2, we introduce the few-shot learning problem and approaches to tackle it. We describe and discuss existing works that are relevant to the subsequent chapters. We then introduce our contribution and position them in the literature.

In chapter 3, we focus on the first stage of few-shot learning methods where we use data from well-represented classes to learn a representation of images. We discuss the choice of model and the training procedure. We also propose a novel method in which the training uses densely computed local descriptors. Using this training method, we observe a better generalization from the large dataset to the few-shot dataset.

In chapter 4, we tackle the question of the possible adaptation of the representation function to the few-shot task by utilizing only the few examples available. We propose two possible solutions. The first one expands the trained network by adding a limited set of parameters trained on the few-shot data. The second consists in adjusting slightly the parameters of the network by few steps of optimization using the few-shot data. Both methods successfully create task-specific representation functions that ultimately result in higher accuracy on the few-shot tasks.

In chapter 5, we study a special case of the few-shot classification problem where multiple images are to be classified at the same time. We propose a model that takes advantage of the availability of other unlabeled images. In particular, we draw a connection between local representations of labeled and unlabeled images. We then propagate the label information to all representations at once. In this context, we also introduce a spatial attention mechanism that filters out irrelevant background.

In chapter 6, we propose a novel few-shot setting to address cases where the in-domain data used to learn the representation in the first stage is few. We propose to use a pre-trained network for the representation function. This network is then adapted to the domain if in-domain data is available before adapting again to the few-shot task using the few examples. In this context, we propose another spatial attention mechanism that takes advantage of the pre-trained classifier.

In chapter 7, we focus on a specific aerial images classification task. We apply methods from the other chapters and discuss their effectiveness in such task.

Chapter 2

Background

Contents

In this chapter we formally introduce the necessary background for the rest of the thesis. First, we formally introduce the problem of few-shot learning. Then we explain the general frameworks that few-shot learning methods follow. We discuss the most influential existing works, focusing on those that are linked to the work of the thesis. We also present our solutions and position them against the literature.

2.1 Fundamental concepts

We consider an image classification problem based on supervised learning. We are given a collection of n *training* examples $X := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ with each $\mathbf{x}_i \in \mathcal{X}$, and corresponding labels $Y := (y_1, \ldots, y_n)$ in c classes. The objective is to build a *classifier* function $f : \mathcal{X} \to \mathbb{R}^c$ that maps the input to class confidence. Then a prediction for input $x \in \mathcal{X}$ is made by assigning the label of maximum confidence, $\arg\max_i f^i(x)$[1].

We aim at finding a model that minimizes the *expected risk* R, that is, the expectancy of the loss over the complete distribution of images P(x,y).

$$R(f) = \int \ell(f(x), y) dP(x, y) = \mathbb{E}[\ell(f(x), y)] \tag{2.1}$$

With $\ell : \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$ a loss function that expresses the discrepancy between a prediction and a label vector such as the *cross-entropy loss*

$$\ell(f(x), y) = -\log(f^y(x)). \tag{2.2}$$

The optimal model is noted f^*. Search of the optimal function is limited by the considered model hypothesis search, we note $\hat{f}$ the function that minimizes the expected risk in the search space. Because the probability distribution of images is unknown, we can only approximate the expected risk with the *empirical risk*

$$R_{X,Y}(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i). \tag{2.3}$$

We note $\tilde{f}$ the function inside the search space that minimize the empirical risk. Assuming we find $\tilde{f}$ through optimization of the parameters on the training dataset. The

[1]Given vector $\mathbf{x} \in \mathbb{R}^m$, x^i denotes the i-th element of $\mathbf{x}$. Similarly for $f : A \to \mathbb{R}^m$, $f^i(a)$ denotes the i-the element of $f(a)$ for $a \in A$.

error with respect to f^* is

$$\mathbb{E}[R(\tilde{f}) - R(f^*)] = \mathbb{E}[R(\hat{f}) - R(f^*)] + \mathbb{E}[R(\tilde{f}) - R(\hat{f})] = \mathcal{E}_{app} + \mathcal{E}_{est}. \tag{2.4}$$

$\mathcal{E}_{app}$ corresponds to the *approximation error* that relates to the ability of the hypothesis space to approximate the optimal solution. $\mathcal{E}_{est}$ corresponds to the *estimation error* which reflects the impact of using the *empirical risk* instead of *the expected risk*.

The *approximation error* depends on the size of the hypothesis space. The larger the hypothesis space is, the smaller is this error. In particular, deep learning models used for state of the art image classification tasks can approximate a wide range of functions [Elb+20], making the *approximate error* theoretically small.

On the other hand *estimation error* grows with the size of the hypothesis space [BB08]. In general, it can be reduced using a large number of examples to train on. However, in the case of few-shot learning the available examples are only few. In this case, empirical risk minimization fails as $\tilde{f}$ is not a good approximation of $\hat{f}$ [Wan+20a]. This phenomenon is called *overfitting*. As a result learning from scratch to solve an image classification on only the few available examples is not feasible.

In order to mitigate those limitations, prior knowledge has to be used in complement to the few examples in the form of extra data. The extra set of data should be large enough to compensate for the lack of supervision for the few-shot task. Depending on the few-shot method, prior knowledge is used differently. In all cases, it is at least used to restrict the size of the hypothesis search which limits the estimation error.

2.2 Few-shot classification problem formulation

We are given a collection of *training* examples $X := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ with each $\mathbf{x}_i \in \mathcal{X}$, and corresponding labels $Y := (y_1, \ldots, y_n)$ with each $y_i \in C$, where $C := [c]$[2] is a set of *base classes*. This training data is prior knowledge that can be used to learn a generic model that will then be used to learn new tasks. The model can be a task agnostic representation function or a training method in the case of meta-learning.

For each few-shot learning task, we are given a collection of few *support* examples $X' := (\mathbf{x}'_1, \ldots, \mathbf{x}'_{n'})$ with each $\mathbf{x}'_i \in \mathcal{X}$, and corresponding labels $Y' := (y'_1, \ldots, y'_{n'})$ with each $y'_i \in C'$, where $C' := [c']$ is a set of *novel classes* disjoint from C and $n' \ll n$. Support examples are used in combination with the generic model learned on base data to perform the few-shot classification task.

[2]We use the notation $[i] := \{1, \ldots, i\}$ for $i \in \mathbb{N}$.

Classification is called c'-*way* where c' is the number of novel classes; in case there is a fixed number k of support examples per novel class, it is called k-*shot*. Typically, few-shot models are evaluated with between one and ten shots. As in standard classification, there is usually a collection of queries for the evaluation of each task. Few-shot learning is typically evaluated on a large number of new tasks, with queries and support examples randomly sampled from (X', Y').

2.3 Frameworks

2.3.1 Traditional framework

In this section, we introduce the standard framework that most non-meta-learning few-shot learning methods follow.

Learning stages We can distinguish two stages of learning. The first stage consists in learning a representation of the domain of images $\mathcal{X}$. This stage uses images from the base classes only. It should be general enough to be used for new tasks. Training is performed by learning to solve a task or multiple tasks built from the base class dataset. We call this stage the *representation learning stage*.

The second stage consists in using the support examples to learn a classifier that maps a new *query* example from $\mathcal{X}$ to a label prediction in C'. It does not exclude continuing representation learning on the support examples. We refer to the latter classifier learning as *adaptation stage*.

Network model We consider a model that is conceptually composed of two parts: an embedding network and a classifier. The *embedding network* $\phi_\theta : \mathcal{X} \to \mathbb{R}^{r \times d}$ maps the input to an embedding, where θ denotes its parameters. Since we shall be studying the spatial properties of the input, the embedding is not a vector but rather a tensor, where r represents the spatial dimensions and d the feature dimensions. For a 2d input image and a convolutional network for instance, the embedding is a 3d tensor in $\mathbb{R}^{w \times h \times d}$ taken as the activation of the last convolutional layer , where $r = w \times h$ is the spatial resolution and $\Omega := [w] \times [h]$ is the spatial domain. The embedding tensor $F := \phi_\theta(\mathbf{x})$ contains a feature vector $F(\mathbf{r}) = \phi_\theta(\mathbf{x})(\mathbf{r}) \in \mathbb{R}^d$ for each spatial position $\mathbf{r} \in \Omega$. In most cases, the spatial resolution is reduced to $r = 1$, transforming the tensor to a vector, using spatial pooling operations such as average pooling or max-pooling.

The *classifier* can be of any form and depends on the particular model, but it is applied on top of ϕ_θ and its output represents confidence over c (resp. c') base (resp.

novel) classes. If we denote by $f_\theta : \mathcal{X} \to \mathbb{R}^c$ (resp. $\mathbb{R}^{c'}$) the *network function* mapping the input to class confidence, then a prediction for input $x \in \mathcal{X}$ is made by assigning the label of maximum confidence, $\arg\max_i f_\theta^i(x)$

2.3.2 Meta-learning framework

Meta-learning refers to models that aim at "learning to learn". This idea comes from the limitations of the machine learning models, one of them being the difficulty of learning with few data. In particular, here, the goal is to learn how to learn with only few examples, which is not possible with usual algorithms. Meta-learning models have been proposed before the rise in popularity of deep learning [Sch87; BBC91] but have recently become popular solutions for few-shot learning, as well as reinforcement learning [FAL17; Wan+17b; XHS18], neural architecture search [Kim+18; Lia+20], unsupervised learning [Met+19; AS19] and other tasks.

Meta-learning models contain two parametric models, the *learner* and the *meta-learner*. The learner learns to perform the task based on a given dataset and a learning algorithm. Its training process is the inner loop of the meta-learning methods. The meta-learner is defining the learning algorithm used for the inner loop. The meta-learner parameters are learned as well, using a loss function that reflects the quality of the optimization of the learner. Parameters that are learned through meta-learning can be for instance, a network initialization [FAL17], a model that generate update rules [RL17], a generator that generates additional data [Wan+18b].

Specifically, in the case of few-shot classification, the meta-learner learns a learning method that allows the learner to learn new tasks with few examples. Therefore, the expected risk of the meta-learner is the expectancy of the trained learner over the few-shot task distribution $p(T)$:

$$R(\theta_{meta}) = \mathbb{E}[\ell(f_T, \theta_{meta})], \tag{2.5}$$

with f_T the learner classifier after training on support data of task T, and θ_{meta} the parameters of the meta-learner. Training is performed through empirical risk minimization by sampling tasks from the base dataset. Tasks can be sampled by sampling a subset of classes and then sampling examples inside those classes. Few examples are used as support in the learner training, some extra examples are used for evaluation of the learner for training the meta-learner.

Learning iteratively on sampled tasks from the base dataset is called *episode learning*. The operation is repeated on many different tasks, while the meta-learner adjusts the

learning algorithm. After meta-learning is completed, a novel task can be learned by application of the training strategy.

In the case of episode learning, there is no methological distinction between what is performed on the base dataset and on the test dataset. For the sake of simplicity, while describing such methods, we name an episode support set X', and corresponding labels Y' with each $y'_i \in C'$, even though when meta-learning, the set is sampled from the larger set X with labels Y in classes C

2.4 Main approaches

This section presents the most influential approaches for few-shot learning, on which our work is based on.

2.4.1 Learning to compare

Solving a few-shot classification problem can be solved by learning to compare images. Indeed, assuming we have access to a comparison function that reflects the semantic similarities between images, for a given query, we can compare it to labeled references for each class, and then aggregate all similarities to form a prediction. References can be all supports examples, as in nearest-neighbor classification [BSI08b], or aggregated example representations [SSZ17]. In this section we present influential methods for few-shot learning where the base class dataset is used to learn explicitly to correctly compare images. We can consider all those methods as *metric learning* methods. Initially, metric learning aims at learning comparison functions between examples, but in practice, it usually focuses on learning a good embedding function and then using a fixed function for comparison in the resulting feature space.

Siamese networks A *siamese network* model learns to compare two examples [Bro+94] by mapping them to a common feature space and learning a comparison function. Koch *et al.* [KZS15] proposed to use this model for one-shot learning. They use a CNN architecture with a fully connected layer on top, which maps an input x to $\phi_\theta(x)$. During training, pairs of images are sampled from the base classes. Two examples from the same class form a positive pair while examples from different classes form a negative pair. Positive pairs are assigned label 1 and negative pairs label 0. The prediction for a

given pair of examples (x_1, x_2) is

$$p := S\left(\sum_i \alpha_i |\phi_\theta^i(x_1) - \phi_\theta^i(x_2)|\right), \tag{2.6}$$

where S is the sigmoid function and α_j are parameters of the comparison function. Learning of the network parameters θ and comparison function α, is performed through minimization of a regularized *cross-entropy* loss function

$$\ell(x_1, x_2) := y(x_1, x_2) \log p(x_1, x_2) + (1 - y(x_1, x_2)) \log(1 - y(x_1, x_2)) + R, \tag{2.7}$$

with R a regularization term. When presented with a novel one-shot problem, the queries are assigned the class for which the support example has maximum similarity.

Relation networks Building on siamese networks, Sung *et al.* [Yan+18] proposed *relation network*. Similarly to meta-learning, training is performed on episodes mimicking few-shot tasks, sampled from the base dataset. However, in this method, no learning happens at episode level since the examples are classified by comparing the query representation with support representations as done in siamese networks [KZS15]. The embedding network is a CNN, so the representations of images are feature maps in $\mathbb{R}^{r \times d}$. For an episode, support examples and the query examples feature maps are computed. When more than one support per class is given (k-shot with $k > 1$), representations of support examples from each class are average to get a unique representation per class. Contrary to siamese networks where comparison between representations is performed by a single layer (2.6), relation networks use a multi-layer *relation module*. The relation module is a CNN as well, it takes as input the concatenation of the query representation and a class representation and generates the relation score. This is repeated for all classes in the episode. Optimization of both the embedding network and the representation module is performed using the mean square error loss with ground truth 1 for the positive class and 0 for the negative class:

$$\ell(\mathbf{x}, \mathbf{y}) := r_{\mathbf{y}}(\mathbf{x}) - 1 + \sum_{i \in C' \setminus \mathbf{y}} r_i(\mathbf{x}), \tag{2.8}$$

with r_i the relation score with class i. Classification decision is done by assigning the class with maximum relation score.

Matching networks Vinyals *et al.* [Vin+16] proposed *matching networks* as a solution for learning the embedding function also based on episode learning. Again, no learning happens at episode level, examples are classified with a non-parametric classifier. In particular, support examples features are extracted by a CNN model ϕ_{sup}. Query examples features are either extracted with ϕ_{query}, which is either the same CNN or a bidirectional long-short term memory [HS97] that combines them with the support examples context. In all cases, a pooling operation reduces the spatial dimension of the output of the network, making the representation vectors in $\mathbb{R}^d$. For a given query examples $\mathbf{x}$ with representation $\phi_{query}(\mathbf{x})$, a similarity score with all support representations $\phi_{sup}(x'_i)$ is computed:

$$a(\mathbf{x}, x'_i) := \frac{exp(\cos(\phi_{query}(\mathbf{x}), \phi_{sup}(x'_i)))}{\sum_j exp(\cos(\phi_{query}(\mathbf{x}), \phi_{sup}(x'_j)))} \tag{2.9}$$

where cos is *cosine similarity*. The network function is the aggregation of the the scores for all support examples

$$f_\theta(x) := \sum_{i=0}^{n'} a(x, x_i) y_i. \tag{2.10}$$

The embedding functions are trained using minimization of the cross-entropy loss.

Naïve Bayes nearest neighbor Li *et al.* proposed a method similar to matching networks in their revival [Li+19b] of the classic image-to-class approach [BSI08a], in that the similarities between query representations and support representations are aggregated to obtain a class score. The main difference is the use of local descriptors of the images instead of global ones. Here, no pooling is performed on the feature maps. Pixels of the feature maps are considered as local representations of the images to be used independently. More formally, given X', Y' and an index set $S \subset N' := [n']$, let the set $S_j := \{i \in S : y'_i = j\}$ index the support examples in S. One collects, for each class $j \in [c']$, the features $V_j := \{\phi_{\theta'}(\mathbf{x}'_i)(\mathbf{r})\}_{i \in S_j, \mathbf{r} \in \Omega}$ of all spatial positions of all support examples labeled in class j. Then, given a query $\mathbf{x} \in \mathcal{X}$ with feature tensor $F := \phi_\theta(\mathbf{x})$, for each class j, a score

$$s_j(F) := \sum_{\mathbf{r} \in \Omega} \sum_{\mathbf{v} \in \mathrm{NN}_{V_j}(F(\mathbf{r}))} \cos(F(\mathbf{r}), \mathbf{v}) \tag{2.11}$$

is defined as the average cosine similarity over the features $F(\mathbf{r})$ at all spatial positions $\mathbf{r} \in \Omega$ and their k-nearest neighbors $\mathrm{NN}_{V_j}(F(\mathbf{r}))$ in V_j. Then, the prediction for $\mathbf{x}$ is the

class of maximum score.

Prototypical networks Another popular episode based learning method, called *prototypical networks*, has been proposed by Snell *et al.* [SSZ17]. While [Vin+16] use similarity measures between a query and each support representations, in prototypical networks, a single vector is used as reference for the each novel class. The reference vector is called *prototype*.

More formally, given S_j an index indexing the support examples from class j. The *prototype* of class j is given by the average of those examples

$$\mathbf{p}_j := \frac{1}{|S_j|} \sum_{i \in S_j} \phi_\theta(\mathbf{x}'_i) \tag{2.12}$$

for $j \in C'$. Then, the network function is defined as [3]

$$f_\theta(\mathbf{x}) := \boldsymbol{\sigma}\left([s(\phi_\theta(\mathbf{x}), \mathbf{p}_j)]_{j=1}^{c'}\right) \tag{2.13}$$

for $\mathbf{x} \in \mathcal{X}$, where $P := (\mathbf{p}_1, \ldots, \mathbf{p}_{c'})$ and s is a similarity function which is generally the negative squared euclidean distance, and $\boldsymbol{\sigma} : \mathbb{R}^m \to \mathbb{R}^m$ is the *softmax function* defined by

$$\boldsymbol{\sigma}(\mathbf{x}) := \left[\frac{\exp(x^j)}{\sum_{i=1}^m \exp(x^i)}\right]_{j=1}^m \tag{2.14}$$

for $\mathbf{x} \in \mathbb{R}^m$ and $m \in \mathbb{N}$.

Graph-base approaches Garci and Bruna showed that few-shot learning tasks can be represented by a graph where each node represents an example [GB18]. Common to other methods, examples are first mapped to embedding using a CNN embedding network. Then a graph with all examples is constructed. Node features of the graph are given by the concatenation of the representations of the examples and the associated one-hot encoded labels. They use a *graph neural network* (GNN) [GMS05; Sca+09] to operate on the graph, computing for each node a prediction on the classification into the novel classes. Graph neural networks consist in multiple layers. In each layer, an adjacency matrix is computed using a parametric comparison function. Then the adjacency matrix is used to aggregate node features, that is, combining features of neighbors in the graph. The aggregation function is also parametric. Both functions'

[3] We define $[e(i)]_{i=1}^n := (e(1), \ldots, e(n))$ for $n \in \mathbb{N}$ and any expression $e(i)$ of variable $i \in \mathbb{N}$.

parameters are learned with backpropagation as in standard neural networks. Garci and Bruna showed that the above few-shot methods can be implemented with GNN. For instance, siamese networks [KZS15] corresponds to a GNN with fixed comparison function. Similarly, prototypical networks [SSZ17] consists in aggregation by averaging of the support examples with the same label, then similarity computation.

An advantage of graph-based methods is the possibility of taking advantage of extra unlabelled data. Any extra unlabelled example can be added to the graph which allows *semi-supervised learning* [GB18] as well as *transductive learning* [Liu+19a]. Transductive learning being the task where multiple queries are to be classified instead of one in the standard setting.

2.4.2 Transfer learning methods

Above methods focus on using the base dataset to learn a representation of images together with a comparison function directly applicable to few-shot tasks. In contrast, transfer learning methods consist in learning a model to solve a *source task* on a large dataset, then the model is adapted for the *target task* using a smaller dataset. Typically, with deep learning based models, after learning the source task, the early layers of the network implement a generic embedding function that is not specific to the source task. Adaptation to the target task is done through fine-tuning of the last few layer of the network [Oqu+14]. Few-shot learning is the extreme case of transfer learning where the task dataset is very small.

Learning with imprinted weights Qi *et al.* [QBL18] proposed a simple approach where the representation learning stage consists in learning to classify into the base classes C. In particular, they use a fully-connected layer without bias as a *parametric linear classifier* on top of the embedding function ϕ_θ followed by softmax and they train in a standard supervised classification setting. More formally, let $\mathbf{w}_j \in \mathbb{R}^{r\times d}$ be the weight parameter of class j for $j \in C$. Then, similarly to (2.13), the network function is defined by

$$f_{\theta,W}(\mathbf{x}) := \boldsymbol{\sigma}\left([s_\tau(\phi_\theta(\mathbf{x}), \mathbf{w}_j)]_{j=1}^{c}\right) \tag{2.15}$$

for $\mathbf{x} \in \mathcal{X}$, where $W := (\mathbf{w}_1, \ldots, \mathbf{w}_c)$ is the collection of class weights and s_τ is the *scaled cosine similarity*

$$s_\tau(\mathbf{x}, \mathbf{y}) := \tau \langle \hat{\mathbf{x}}, \hat{\mathbf{y}} \rangle \tag{2.16}$$

for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{r \times d}$; $\hat{\mathbf{x}} := \mathbf{x} / \|\mathbf{x}\|$ is the ℓ_2-normalized counterpart of $\mathbf{x}$ for $\mathbf{x} \in \mathbb{R}^{r \times d}$; $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ denote Frobenius inner product and norm respectively; and $\tau \in \mathbb{R}^+$ is a trainable scale parameter. Then, training amounts to minimizing over θ, W the *cost function*

$$J(X, Y; \theta, W) := \sum_{i=1}^{n} \ell(f_{\theta,W}(\mathbf{x}_i), y_i) \tag{2.17}$$

where ℓ is the *cross-entropy* loss.

Given a new task with support data (X', Y') over novel classes C', class prototypes P are computed on N' according to (2.12). They can replace class weights W in the classifier, in which case it is equivalent to using a prototype classifier, with scaled cosine similarity as similarity function. Alternatively, prototypes can be *imprinted* in the classifier, that is, W is replaced by $W' := (W, P)$. The network can now make predictions on $n + n'$ base and novel classes.

Few-shot learning without forgetting Gidaris and Komodakis [GK18], concurrently with [QBL18], developed a similar model that is able to classify examples of both base and novel classes. The main difference to [QBL18] is that they learn a *few-shot classification weight generator* module to generate novel classes weights, rather than using prototypes. The weight generator is a parametric classifier that combines support representations and base class weights. Training of the weight generator requires an additional training phase and is achieved through episode-based learning.

Comparison with learning to compare approaches Where learning to compare approaches explicitly enforce good comparison in in the feature space, in the case of transfer learning methods, the embedding function is implicitly learned on a disjoint multiclass classification problem. One might expect that the former models result in higher performance on the few-shot tasks. However, surprisingly, the latter models tend to perform as well or better as shown in [Che+19; Wan+19]. In particular, Wang *et al.* proposed *SimpleShot* [Wan+19], a simple model similar to [QBL18], except for the choice of inverse squared euclidean distance as similarity function and the centering of the feature representation. Their extensive experiments show impressive performance compared to more complex meta-learning methods.

2.4.3 Adapting the representation to the few-shot task

The first stage of learning uses the the base dataset to learn a good representation of the images, in some cases together with a generic way of comparing data. An issue of reusing the representation on a new few-shot task is the lack of adaptation to the available support examples. Even if this data is few, using it to create a task-specific model is appealing. The intuitive approach is to have an additional training stage on the support examples. Fine-tuning of the last few layers of the network is typically performed in transfer learning methods [Oqu+14]. The risk here is to overfit to the support examples, which would result in worse generalization on the queries. Here we present relevant methods for adapting representation functions.

Base and novel classes fine-tuning For methods that allow classification into both base and novel classes [QBL18; GK18], it is possible to fine-tune the model using (2.17) on the union of the base dataset and the support examples. In this case a class balancing method should be used. For instance, [QBL18] oversamples the novel classes so that the distribution of labels in a sampled training batch is uniform across all classes. This method is specific to the problem of classification into all classes. Moreover, it requires to have access to the base dataset to perform a new expensive training stage for every few-shot task, which is not standard.

MAML Finn *et al.* proposed model agnostic meta-learning (MAML) [FAL17], a meta-learning method in which the meta-learner is learning a good initialization for fast adaptation of the model on the support examples. It uses a fully-connected layer as classifier. At adaptation, the embedding function is fine-tuned for each new task using (2.17) only on the novel class data, but for *few steps* such that the classifier does not overfit. In [FAL17], the whole embedding network is fine-tuned, while in [Sun+19], fine-tuning is limited a scale and shifting parameters at every layers to further prevent overfitting.

More formally, if we consider a learner on task T that performs a single gradient update of size α, the parameters θ of the learner becomes θ'

$$\theta' = \theta - \alpha \nabla_\theta \ell_T(f_\theta), \tag{2.18}$$

with ℓ_T the cross-entropy loss computed on the support set of T. During meta-learning, the initial parameters θ are updated based on the loss of multiple learners that learned

with task T_i sampled from the task distribution $p(T)$

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \ell_{T_i}(f'_\theta), \tag{2.19}$$

with β the size of the meta-learner update. Therefore, the meta-learning stage of MAML requires to compute second derivatives which is computationally expensive. A first order approximation can be used instead to solve this issue [FAL17; NAS18] without loss of performance.

Meta-Learner LSTM Ravi and Larochelle go a step further into the conditioning of the learner training by the meta-learner. They observe that traditional optimization algorithm like *stochastic gradient descent* (SGD) is not suited for few-shot learning. In [RL17], they propose to train an LSTM [HS97] to predict a suitable update rule for the parameters. The gradient update (2.18) is replaced by

$$\theta \leftarrow g \odot \theta - i \odot \nabla_\theta \ell_{T_i}(f_\theta) \tag{2.20}$$

where g and i are parametric are the output of parametric regressors with input their previous state and $(\nabla_\theta \ell_{T_i}(f_\theta)$, $\ell_{T_i}(f_\theta), \theta$ and $\odot$ is the Hadamard product. g not being fixed to an all-one vector allows fast forgetting of the parameters when the loss is high even with low gradients. i replaces the fixed step size by a learned one. Meta-learning parameters related to the the update rule are shared across all parameters of the network for computation cost and storage reasons.

Learning to predict layer parameters Some methods propose to obtain task-specific embedding functions by predicting some of their parameters based on the support examples without actually training on them. In an early one-shot learning work [Ber+16], Bertinetto *et al.* propose to learn a model that takes as input a support example and predicts parameters of layers of the embedding network. Because of the large number of weights of neural networks layers, it is impossible to naively predict all of them. Instead, they design factoring methods for both convolutional layers and linear layers to limit the output size of the predictor. Parameters of the predictor as well as the fixed parameters of the embedding networks are learned with an episodic training strategy.

TADAM Similarly, Oreshkin *et al.* [OLR18] proposed task-dependent adaptive metric (TADAM), which builds on prototypical networks [SSZ17] by introducing task con-

ditioning, that is, including task-specific parameters in the embedding network. More specifically, the mean of the prototypes are used as representation of the task. This representation is input to *task embedding networks* (TEN) which predict, for each layer of the embedding network, additional scaling and shifting parameters γ and α. Given a specific layer with corresponding representation of examples h, a new operation is added:

$$h \leftarrow \gamma \odot h + \beta, \tag{2.21}$$

γ and β having as dimension the depth of the h. The modified embedding network is then used as in prototypical networks. The parameters of the TEN are learned during episodic learning, at the same time as the embedding network.

2.5 Data augmentation

Data augmentation is a common preprocessing stage when training a model to classify images. It usually consists in applying fixed transformations on images such as flipping, rotation, scaling, cropping, color jitter, effectively multiplying the number of training examples. This augmentation makes the model invariant to such transformations, resulting in better generalization. In the case of few-shot learning, such an augmentation strategy is insufficient as the main obstacle is that the lack of diversity of annotated examples which those methods cannot fix. More complex methods have been proposed to artificially augment the size of the support set.

Hallucination *Hallucination* refers to the process of applying transformations on an example to generate another of the same class. In few-shot learning, we are interested in applying such process on the support examples. In an early one-shot learning work, Miller *et al.* proposed to learn a distribution of 2D transforms that models intra-class variability in handwritten digits classes [MMV00]. When presented with a new character, it can be augmented by sampling transforms in the learned distribution. For most domains outside of characters, 2D transforms cannot capture the complex relationship between different examples in a class. Researchers produced solutions for hallucination based on combining examples from the base classes and examples from the novel classes to produce new relevant ones.

Hariharan and Girshick proposed to use a multilayer perception as generator for hallucination [HG17]. Their idea is to capture analogies between examples from the same classes in the base dataset. For instance, in the case of bird species categories, the

relationship between a bird sitting on a branch and the same bird flying is transferable to other bird classes. The first step of the method is to mine analogies. For each base class, they cluster the representations of all examples into a fixed number of clusters. For each pair of cluster centroids $(c_{a,1}, c_{a,2})$ in class a, they search pairs from other classes that correspond. Pair $(c_{b,1}, c_{b,2})$ from class b matches if $\cos(c_{a,1} - c_{a,2}, c_{b,1} - c_{b,2}) > 0$. All gathered quadruplets of the form $(c_{a,1}, c_{a,2}, c_{b,1}, c_{b,2})$ are used as training set for the generator. It takes as input $(c_{a,1}, c_{b,1}, c_{b,2})$ and output a prediction $\hat{c_{a,2}}$ of $c_{a,2}$. The training loss is a combination between the mean squared error function between $c_{a,2}$ and $\hat{c_{a,2}}$ and the classification loss that results in applying the classifier on $\hat{c_{a,2}}$ where a is the ground-truth label. After training, a new example of the novel classes can be generated from a support example representation $\phi_\theta(x')$ by applying the generator on $(\phi_\theta(x'), c_{a,1}, c_{a,2})$ with the pair $(c_{a,1}, c_{a,2})$ randomly sampled from the mined pairs.

Similarly Schwarz and Karlinsky proposed to use a modified auto-encoder called Δ-encoder that takes as input a base class representation a novel classes representation, and learn to adapt the base class example to the novel class [Sch+18]. Liu *et al.* go one step further by performing class-to-class translation in the image space [Liu+19a].

Wang *et al.* proposed to train an hallucinator network with meta-learning. Taking inspiration from the recent generation literature [Goo+14; KW14], the hallucinator takes as input an example representation and a random noise vector. The output is used as additional support, forming a larger support set. Training is performed conjointly with an episode based method such as matching networks [Vin+16] or prototypical networks [SSZ17].

Instead of hallucinating new examples, Afrasiyabi *et al.* proposed to reuse part of the base class examples when learning to classify on the novel classes [ALG19]. For each novel class, the closest base classes are selected. For a few-shot task, the embedding network is fine-tuned similarly to [QBL18]. An additional loss term is added to enforce alignment of the support representation and representations of examples from related base classes.

Using extra unlabeled data Some works propose to depart from the initial few-shot learning setting to take advantage of extra unlabeled data. Ren *et al.* defined a new semi-supervised setting where on top of support examples, a set of unlabeled images, some of which being relevant for the task, are available [Ren+18b]. They proposed a modification of prototypical networks [SSZ17] where the prototype computation (2.12) is replaced by a weighted average of the representations of the support and the extra examples representations. Weights for the extra examples are computed as a function

of the distance to the original prototypes (2.12). A distractor class is also introduced to prevent unrelated images from affecting the prototype classifier.

Using extra unlabeled data is straightforward with graph-based methods [GB18; Liu+19a]. Any extra unlabelled example can be added to the graph. Propagation of label can also label the unlabelled data, making it a solution for *transductive few-shot learning*. Transductive few-shot learning being the task where multiple queries are to be classified instead of one in the standard setting. Douze *et al.* employ a similar method but take advantage of a very large-scale dataset of almost 100 millions images [Dou+18]. Iscen *et al.* [Isc+19b] used the same large-scale dataset, but applied filtering based on the names of the novel classes. Filtered examples are given a relevancy scores by a GNN and are then used as noisy data for learning a prototype classifier or a cosine classifier.

2.6 Boosting few-shot learning

One of the key aspect of a successful few-shot learning method is to be able to learn a robust embedding function. In this section we present methods that apply generalization boosting methods which have been proven useful for other tasks, in the context of few-shot learning. Boosting methods are applied in conjunction with a few-shot learning method as introduced above.

Ensemble of models Quality of prediction can be improved by using an *ensemble* strategy [HTF09]. Instead of training a single model, a set of models is used, predictions are then aggregated to provide a single prediction which usually results in lower variance and in some cases higher average accuracy. Dvornik *et al.* proposed to use an ensemble of models for few-shot learning [DSM19]. Each model instance is trained like prototypical networks [SSZ17]. Ensemble models can benefit from the diversity of models predictions. Diversity is encouraged by adding a specific term to the loss function. Specifically, given two models output predictions p_1 and p_2 where predictions for the ground truth class are put to 0, then $\cos(p_1, p_2)$ is a possible penalty. This penalty is minimal when predictions for incorrect classes are orthogonal. At the same time, cooperation of models results in faster and more stable training which is important with small ensembles. Cooperation can be encouraged using the Kullback–Leibler divergence between p_1 and p_2 are an extra loss term.

Strong performances from ensemble models come at the cost of multiplying the number of parameters and computing cost. In [DSM19], the ensemble model is used as a teacher model in order to train a single model using knowledge distillation [HVD15].

Namely, the average prediction output is used to guide learning of the single model.

Representation learning with additional unsupervised supervision Improving the representation learning can be achieved through extra supervision. In particular, regularization methods are commonly used in classification tasks [Sri+14; IS15; Ver+19]. Mangla *et al.* show that using *manifold mixup* [Man+20] during base class learning results in a more generalizable feature space, ultimately resulting is higher few-shot accuracy. Manifold mixup consists in adding a supplementary loss in the cost function (2.17). Intermediate representation of examples are mixed together by linear interpolation, and their labels are combined in the same way. The additional loss corresponds to to the cross-entropy loss applied on the prediction of the interpolation. Enforcing gradual prediction results in smoother decision boundaries in the intermediate representation spaces.

Additionally *self-supervision* methods can be utilized to learn a richer representation of images. Self-supervision consists in learning to solve auxiliary tasks on the input images. Tasks are defined so that supervision is free of manual annotation. Possible self-supervision tasks include prediction of a random rotation that have been applied in images [GSK18], solving a jigsaw puzzle [NF16] or inpainting [Kra+16]. The assumption is that those tasks help in learning generic features, useful for classification into all classes. In the context of few-shot learning, improved results have been observed by learning to predict image rotations [Gid+19; Man+20], learning to predict relative patch location [Gid+19] and learning to group together representations of augmented versions of examples [Man+20].

2.7 Few-shot learning datasets

In this section we present the few-shot classification datasets that we use in this thesis. Those datasets or variations of them are the most used in recent few-shot learning works. Datasets used for standard classification are usually split into three subsets of images. A portion of images from each class is dedicated to training the model, another is dedicated to validating the parameters and the rest of images are used to test the model. In the case of few-shot classification, the splits of the dataset are performed on the classes. Some classes are dedicated to base classes, some for validation and some for testing. Even if they are to be used for few-shot, classes from the validation and testing set are not few. At validation and test time, many few-shot learning tasks are sampled from those sets to get statistically representative performance results.

2.7.1 Omniglot

Early few-shot learning works [Lak+11; Vin+16; San+16; FAL17; SSZ17] were using the Omniglot dataset [LST15]. This dataset contains 1623 hand-written characters from 50 alphabets, with 20 examples of each character. Images are black and white, with resolution 105×105 but often resized to smaller dimensions. Rapidly, this dataset has proven to be too easy, with simple methods such as prototypical networks [SSZ17] giving 99% average accuracy on 5-way 1-shot tasks even with small neural networks as embedding networks. Representation of such simple images was not a challenge, which can explain why it was not the focus of early few-shot learning works.

2.7.2 *Mini*ImageNet

*Mini*Imagenet has become the most popular few-shot learning dataset. It was introduced by Vinyals *et al.* [Vin+16] as a subset of the larger ImageNet ILSVRC-12 dataset [Rus+14]. It contains 100 classes with 600 images per class. *Mini*ImageNet classes cover a wide diversity of content, containing some animal species classes, as well as vehicles and clothes. While multiple splits have been proposed, the most common is from Ravi and Larochelle [RL17]: 64 classes are used as base classes and 36 as novel, out of which 16 for validation and 20 for testing. The original ImageNet dataset images have large resolutions of various size. Few-shot learning works tend to use a resized version of the images, which is usually 84×84 for older works or larger in more recent ones.

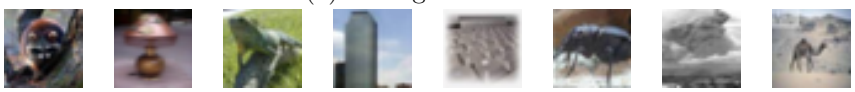
2.7.3 FC100

Oreshkin *et al.* [OLR18] introduced FC100, a few-shot version of CIFAR-100 [Kri09]. Similarily to *mini*ImageNet, CIFAR-100 has 100 classes of 600 images each, although the resolution is 32×32. All classes are grouped into 20 super-classes, for instance "dolphin" and "seal" are grouped into the aquatic mammals superclass. 60 classes are used for training, 20 for validation, and 20 for testing. Those splits are made so that the super classes are not separated, so the classes are more similar in each split, creating a semantic gap between base and novel classes. Because of the small resolution and the semantic gap, FC100 is a more challenging task than *mini*ImageNet.

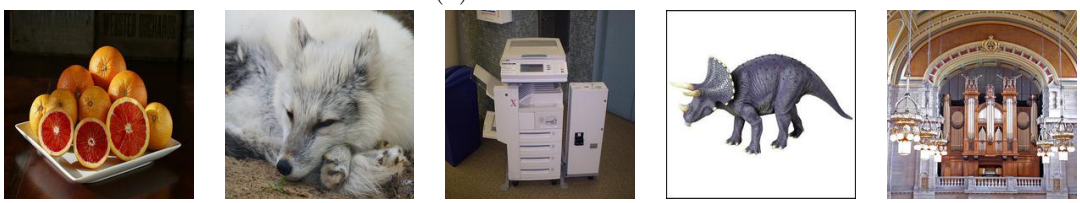
2.7.4 CUB

The topic of fine-grained classification has also been studied in a few-shot learning context. Hilliard *et al.* [Hil+18] have proposed to use the CUB-200-2011 dataset [Wah+11]

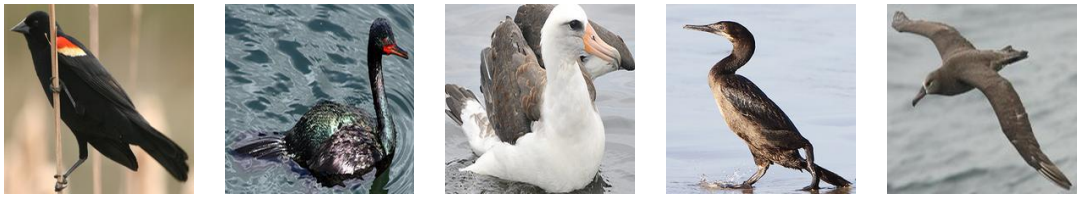
(a) Omniglot dataset



(b) FC100 dataset



(c) *mini*ImageNet dataset



(d) CUB dataset

Figure 2.1 – Examples of images from the few-shot datasets. Each image is from a separate class to illustrate the inter-class diversity of datasets.

for few-shot learning. This dataset contains 11,788 images of birds across 200 classes corresponding to different species. A commonly used split is the one of Ye *et al.* [Ye+18], where 100 classes are used as base classes and the remaining 100 as novel, out of which 50 for validation and 50 for testing. Depending on the work, some preprocessing can be applied on the images such as cropping using bounding box annotations, and resampling.

2.8 Proposed pipeline

In this thesis, we use a simple pipeline where a query is mapped to a representation using an embedding network, then a classifier maps this representation to class predictions. The pipeline is shown in Figure 2.2. We propose to add a spatial attention operation on the representation to select relevant local regions in the image. The embedding function

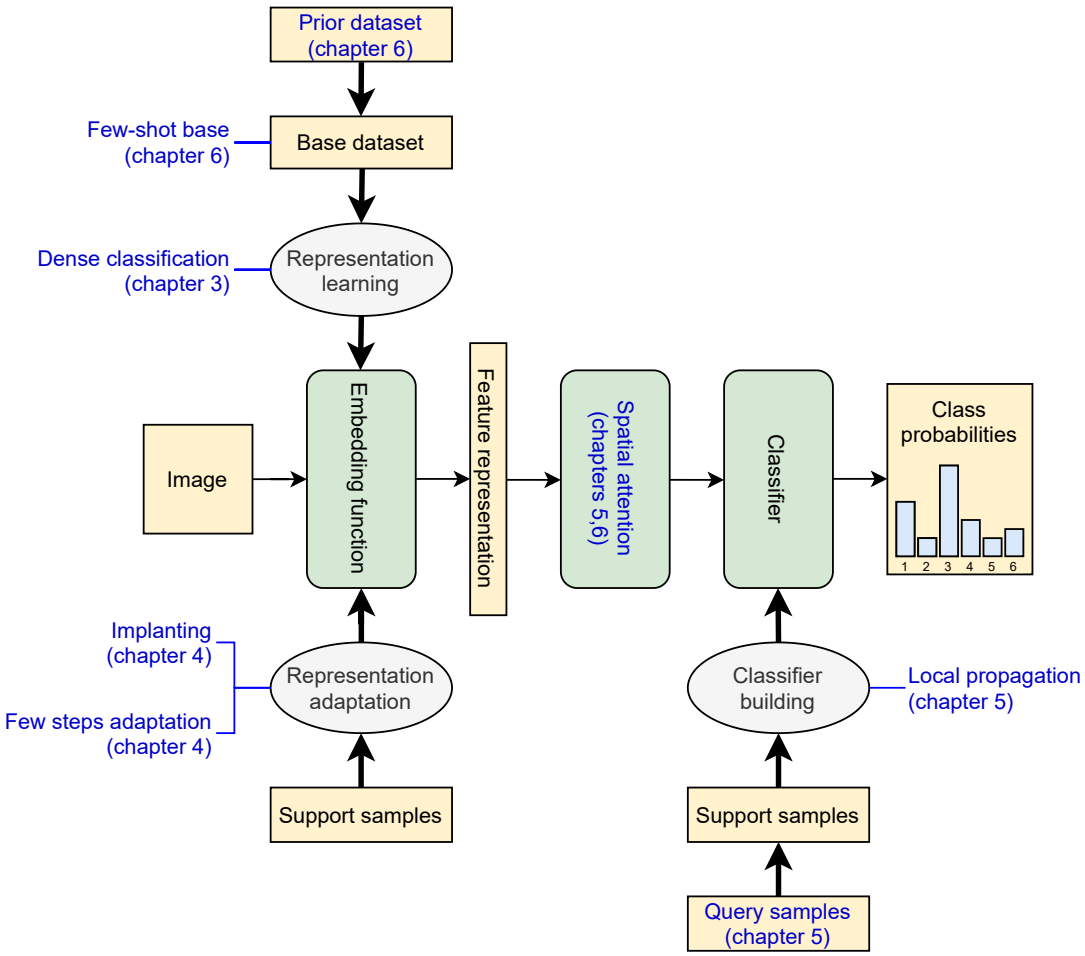


Figure 2.2 – Proposed pipeline for few-shot classification. The feature extractor is trained during representation learning on a prior dataset and then adapted on the support examples. A classifier for the few-shot classes is built from the support examples. A spatial attention module is used to select relevant local representations. In blue are our contributions to the pipeline with the chapter number where they are discussed.

is learned on the base dataset using our proposed method, *dense classification*. We also explore a new setting where we use a prior dataset to pretrain the network, then continue training on few base examples only. We also propose to adapt the embedding function to the few-shot task, using the support examples. In the standard setting, the classifier is built from the support examples. We also study transductive few-shot learning, where multiple queries are to be classified at the same time. Below, we describe our contributions in more detail.

2.9 Positioning

In this section we position our work in the related literature. At the beginning of the thesis, we experimented with popular few-shot learning models at the time, such as prototypical networks [SSZ17], matching networks [Vin+16] and cosine classifier [QBL18; GK18]. In parallel to [Che+19; Wan+19], we observed that methods that were explicitly learning to compare examples by adopting an episode-based learning strategy were not performing better than a simple cosine classifier trained on multiclass classification on the base classes, combined with a prototype-like classifier at test time. Those experiments also revealed the uttermost importance of the choice of architecture to implement the embedding network. From those observations, we decided to focus on improving the representation of images rather than designing complex meta-learning models. We chose to use the cosine classifier model as base learning strategy for representation learning.

2.9.1 Local approach

In this context, in parallel to [Li+19a], we study for the first time spatially local information in chapter 3. In our work, the pixels of feature maps are used as separate local representations of the input example. With our proposed *dense classification* method, classification of all local representations is encouraged, resulting in a more expressive embedding function. Moreover, the distinction between local representations allows us to design simple attention mechanisms in chapter 5 and chapter 6, able to select relevant information in few-shot examples.

2.9.2 Embedding adapation

Our work is also related to methods that aim at producing task-dependent models. More precisely, similarly to MAML [FAL17], we propose in chapter 4 to fine-tune the model on the few support examples alone. However, our method does not rely on meta-

learning. We also propose *implanting*, that is, adding a limited number of task specific parameters to the model. Only the added parameters are trained on the few support examples, reducing the risk of overfitting. This method is related to the work of Sun *et al.* [Sun+19] developed in parallel to ours, in which scaling and shifting parameters for each layer of the embdedding network are trained on the support examples.

2.9.3 Transductive few-shot learning

For transductive few-shot learning, we propose in chapter 5 *local propagation* as a solution to take advantage of the extra unalbeled data. Similarly to transductive propagation networks [Liu+19c], we build a graph with labeled and unlabeled representations and then propagate labels on it. However in our case, the graph is computed using local representations of images, and we introduce a feature propagation step. Although transductive inference has a long history of research, this is the first time label propagation is used for classification.

2.9.4 Using extra unlabeled data

In chapter 6, we redefine the few-shot learning problem, introducing the *few-shot few-shot* setting. The access to the base class data is limited to a few examples. Moreover, we take advantage of prior knowledge in the form of a pre-trained classifier on a large-scale dataset from another domain. Using an extra large scale data has been studied in [Dou+18; Isc+19b], however in those cases, extra data is used for enriching the support set whereas we use it to to build the embedding network.

Chapter 3

Representation learning for few-shot learning

Contents

The image space does not allow semantic separation of images which we would need to solve a classification task. Therefore, we need to convert the images to representations which are representative of their semantic. The machine learning approach to to solve this problem is to learn this representation as a function ϕ_θ whose parameters are learned on relevant data. In a usual classification problem, we would have access to many labeled images coming from the classes that we want to classify. In this case, learning

the representation and learning a classifier for those classes can be done conjointly. A classifier being a model that assigns a class prediction to an image based on its representation. Typically, the representation task would be implemented as a convolutional network to which would be added a fully-connected network to handle the classification. This would allow end-to-end training of both modules. The resulting embedding function would then be optimized to maximize the performance of the classifier, giving the best representation for the task at hand.

In the case of few-shot learning, labeled data from the novel classes, that is to say, classes into which we want to classify are only few. Therefore, it is not enough to hope to learn an effective representation of images. However the few-shot learning scenario assumes we have at disposal a larger collection of images from a semantically similar but disjoint set of classes, the base classes. This set is sufficient in size and diversity to learn the representation function. To do that few-shot learning models introduce intermediate tasks involving base class images. Some few-shot learning methods learn to solve a classification problems over all the base classes. Other methods, based on meta-learning [SSZ17; RL17; FAL17; Mis+18], sample few-shot learning tasks from the base class data to simulate a few-shot setting. Representation learning being the mandatory first step of any few-shot learning method, for simplicity, we call it stage 1 in this chapter. Adapting the representation and using it for classification is referred to as stage 2.

In this chapter, we explore multiple ways of getting a representation function from the base class dataset. We show that the performance of such model depends on the chosen embedding network, the representation learning model and the implementation procedure. From this observation, we select the setting that will be used for the rest of this work as baseline model.

Most few-shot learning approaches do not deal explicitly with spatial information since feature maps are usually flattened or pooled before the classification layer. We propose our own method, called *dense classification*, based on the chosen baseline. With dense classification, we learn to represent and classify regions of the base images, which results in learning a representation function that performs better on novel class data. We show the effectiveness of dense classification through qualitative and quantitative results. This work has been published in [Lif+19]

3.1 Choice of baseline model

In this thesis, we focus on learning representation by learning to solve a classification problem over the base classes. This section motivates this choice as well as implementation choices (network architectures, optimization procedure) for representation learning that were made for the remaining of this manuscript.

3.1.1 Few-shot learning embedding architectures

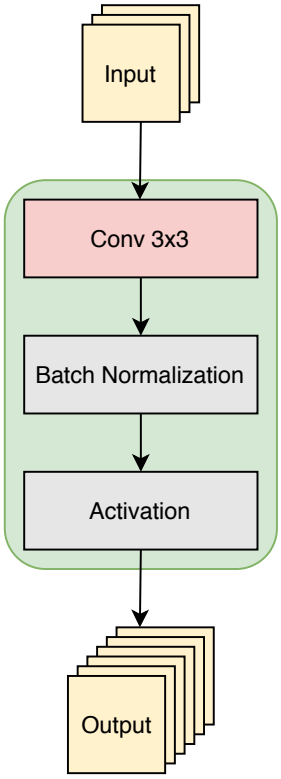
A common trend in image classification tasks is to progressively use deeper networks [SZ14; Sze+15] which usually perform better. Having progressively deeper networks results in a more difficult learning process. For this reason, the networks used to be limited to a few layers [BSF94]. However recent models have overcome optimization issues and use deep and wide architectures, sometimes hundreds of layers [SGS15; He+16].

The same trend applies to few-shot learning models, we observe that the networks used in the literature shifted in the last few years from small four-layer convolutional networks to deep Residual Networks architectures [He+16] and more recently to Wide Residual Network architectures [ZK16].

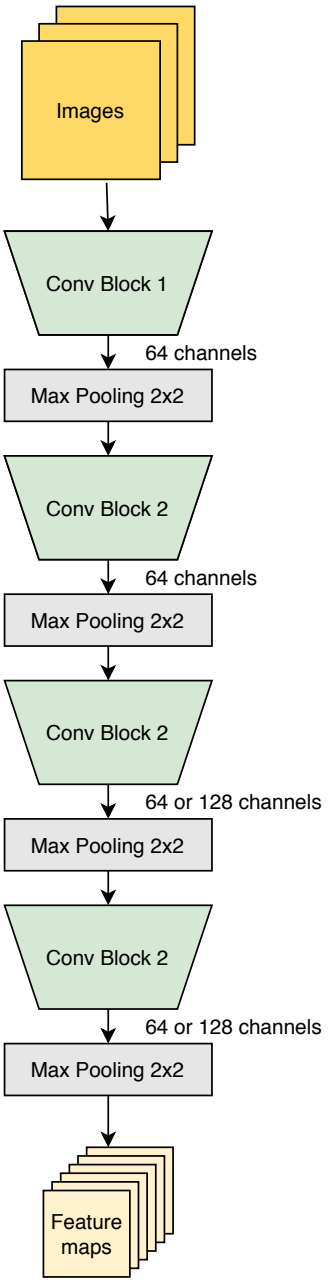
Finding the more effective network architecture for a given task is a research subject of its own [EMH20]. Moreover as we will show, the choice of embedding network is very impactful on the final few-shot performance, therefore to be able to make meaningful comparisons with other methods, using the same or similar embedding network is required. For those reasons we choose to experiment only with common architectures.

In this section we present the most commonly used embedding networks in the few-shot learning community and then motivate the choices we made in our work.

CNN architectures In a pioneer few-shot learning work [Vin+16] is introduced a four layers embedding network, later referenced as C64F. It is composed of 4 convolutional layers, each with 64 filters of size 3×3, batch normalization [IS15], ReLU and maxpooling 2×2. This network was then widely used in other few-shot learning works [FAL17; SSZ17; Yan+18]. A wider version of this network, called C128F, is also used in the literature, with only difference with C64F that the last two layers have 128 channels. Figure 3.1 illustrates those architectures. Those architectures gave impressive results on the Omniglot dataset but our experiments suggest that their expressive power is not sufficient to deal with more complex datasets such as *mini*ImageNet.



(a) Convolution block



(b) 4 layers embedding network architecture

Figure 3.1 – Four layers embedding networks architectures used for few-shot learning. All C64F filters have 64 channels, whereas the two last layers of C128F have 128 channels. The activation function used is ReLU.

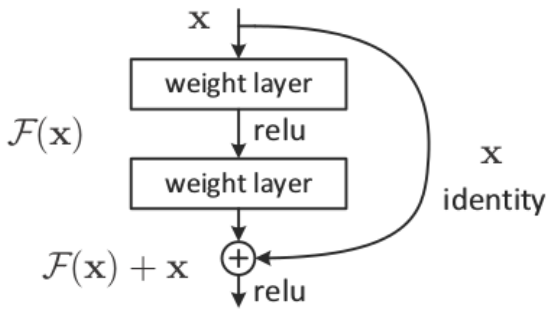
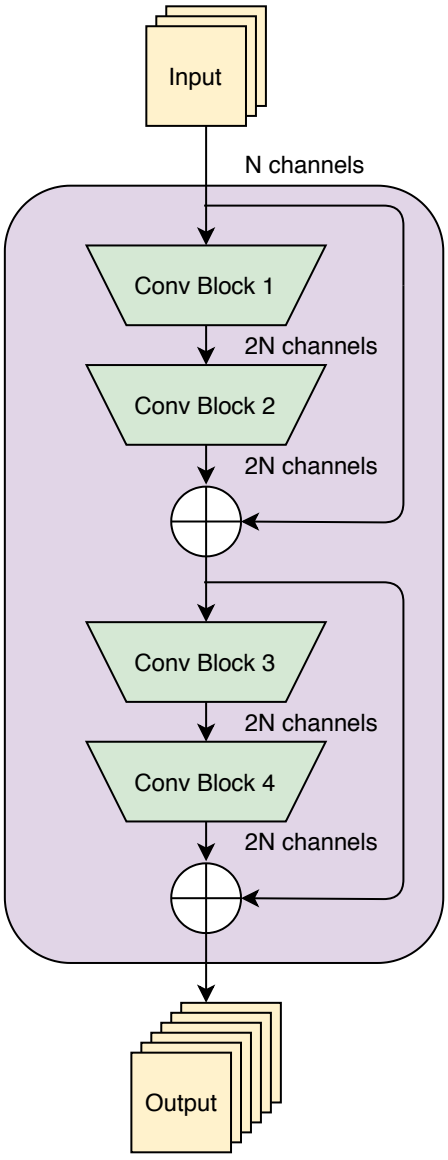


Figure 3.2 – Example of residual block from a residual network [He+16]

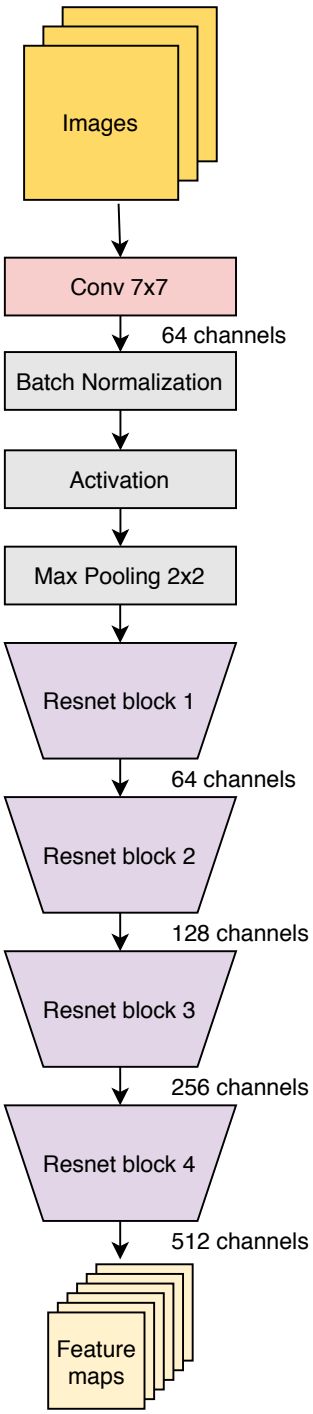
Residual networks architectures To optimize a neural network, the gradient of the loss with respect to all parameters is computed using the backpropagation method. Using the chain rule, the derivative of the first layers is computed as a product of the derivative from deeper layers. If the activation function has lower than 1 derivative, which was always the case using the popular sigmoid activation, the gradients for a given parameter would decrease exponentially with the layer distance to the output of the network, making the training of early layers very challenging. This phenomenon is known as the vanishing gradient problem [BSF94]. This problem, together with a saturation of performance highlighted by some works. [SGS15; He+16], used to limit networks depth. He *et al.* [He+16] proposed a new neural network architecture called Residual network, referred to as ResNet. They propose to add residual connections to convolutional networks. Residual connections are alternative data streams that allow intermediate representations to skip some convolution layers as illustrated in Figure 3.2. This innovation solves the vanishing gradient problem as it creates direct streams of information to deep layers of the network, through which the gradient can propagate without shrinking. Moreover, they showed improved performance on the classification task using very deep architectures.

Multiple works highlighted that the shallow four-layer convolutional networks might not be sufficient with complex images as found in the *mini*ImageNet dataset [Mis+18; OLR18]. Therefore, they proposed to use Residual network architectures, to get an embedding network with higher representative power.

A commonly used architecture is a 18 layers Residual Network called ResNet-18. This architecture also has the advantage to be widely used outside of few-shot learning studies, so its implementation is easily found and some pre-trained version are publically available, of which we will make use in chapter chapter 6. Figure 3.4 illustrates this architecture.



(a) Residual block of ResNet-18. Conv Block refers to the same convolutional block used in C64F and C128F (Figure 3.1a).



(b) Resnet-18 architecture

Figure 3.3 – ResNet-18 architecture. The activation function used is ReLU.

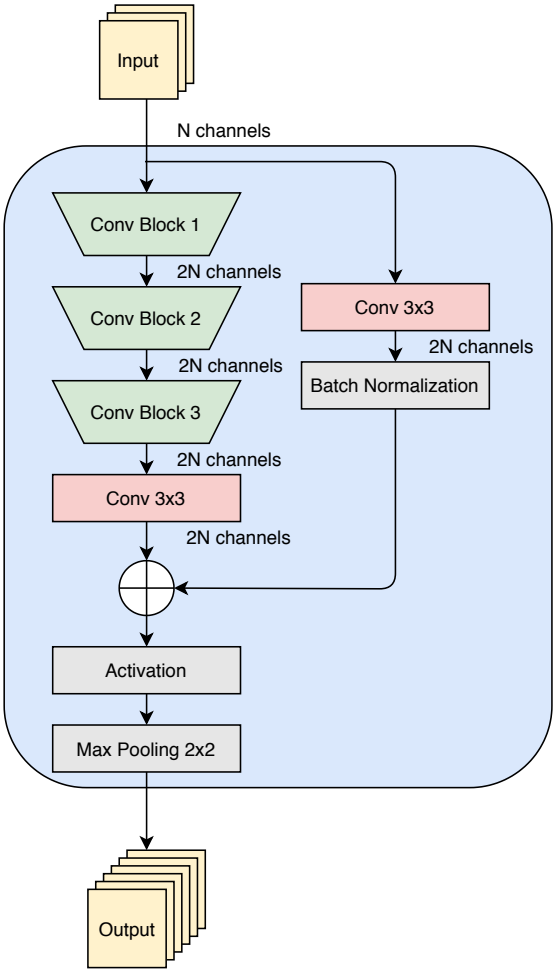
Another common architecture is ResNet-12, introduced by Oreshkin et al. [OLR18], it is composed of four residual blocks, each having three 3×3 convolutional layers with batch normalization [IS15] and swish-1 activation function [RZL18]. Other versions of this network with different activation functions have been used as well [Lee+19] but we chose to use the original implementation. Each block is followed by 2×2 max-pooling. The shortcut connections have a convolutional layer to adapt to the right number of channels. The first block has 64 channels, which is doubled at each subsequent block such that the output has depth 512. Figure 3.4 illustrates this architecture. Oreshkin at al. [OLR18] performed hyper-parameters search to achieve this architecture (applying their method using residual networks with varying number of layers and layer widths). While this network is shallower than ResNet-18, it is worth noting that it has more parameters than the later (about 12 millions, compared to 11 millions for ResNet-18). Afterward, this architecture became a standard in the few-shot learning community.

Wide Residual Networks Zagoruyko et al. [ZK16] highlighted that the residual networks as introduced in [He+16] were not fully benefiting from their sometimes very large depths. Increasing performance by less than a percent of accuracy on a well studied task such as classification on CIFAR-10 would necessitate more than double the number of layers, which makes the optimization very slow. They identified this behaviour as being a consequence of the diminushing feature reuse problem [SGS15], that is to say, nothing forces the network to learn to use the weights of residual blocks, so many of them might have very little contribution to the model. They show that wider architectures, that is to say with more channels, make the use of additional layers more effective.

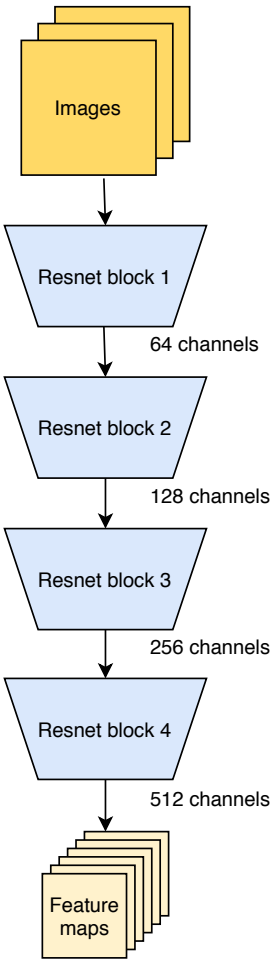
Recently, some few-shot learning papers [Rus+18; Gid+19; Man+20] have turned to a wide residual network architecture called WRN-28-10. This network is composed of 28 layers, each with residual blocks that are 10 times wider than the basic residual network architecture. Because this architecture has only been recently embraced by the few-shot learning community, we did not use it in this work to keep the focus on fair comparison to earlier works. However, it should be noted that using this architecture brings impressive results on common few-shot learning benchmarks [Man+20].

3.1.2 Training procedure

Training a machine learning model consists in solving an optimization problem, that is to say finding the model parameters that minimize a loss function. The loss function is representative of the effectiveness of the model. In the case of a model that is trained



(a) Residual block of ResNet-12. Conv Block refers to the same convolutional block used in C64F and C128F (Figure 3.1a).



(b) Resnet-12 architecture

Figure 3.4 – ResNet-12 architecture. The activation function used is Swish.

to classify images, an example of loss function we can use is the cross-entropy (2.2). It is computed on the training dataset. However, this loss function does not always reflect high performance on other data. In case of overfitting, the model can perform perfectly on training data but without generalizing to unseen data. Therefore, a validation set composed of extra data is generally used to select hyperparameters of the training process, which ensures generalization outside of the training data. In case of few-shot learning, the test classes are disjoint from the base classes, therefore the model must generalize to data outside of the training classes. Stage 1 of few-shot learning models aims at solving a classification on the base classes. Thus, we are presented with two choices of validation metrics. On the one hand, we can build a validation set of images coming from the base classes, which is consistent with the training task. On the other hand, we can build a validation set of images from classes outside of the training classes (and outside of the test classes as well for fairness), which would be consistent with the test setting. In this section, we explore the relationship between the validation measurements that could come from those sets and the impact of choosing one over the other.

Base class validation Because our representation learning stage is equivalent to solving a classification problem over the base classes, it would be natural to use a validation set composed of images from the base classes C and use a traditional classification loss or classification accuracy on it. Concretely, it would mean using a collection of examples $X^{val} := (x_1^{val}, \ldots, x_n^{val})$ with each $x_i^{val} \in \mathcal{X}$, and corresponding labels $Y^{val} := (y_1^{val}, \ldots, y_n^{val})$ with each $y_i^{val} \in C$. In practice, such validation is not always available for the given few-shot learning dataset. For instance, CUB, CIFAR-100 are lacking additional images from the base classes to construct such set. *Mini*ImageNet is a subset of 100 classes of ImageNet, 64 being used as base classes, with 600 examples per class. Because the chosen ImageNet classes have more examples, a disjoint validation and test set using the same 64 classes can be created. We chose for this experiment to use the splits of Gidaris et al. [GK18] from the extra ImageNet data. For each training class, a set of 300 images disjoint from the training images are selected. Assuming we can access such collection of data, we use the classification accuracy of our model on it and we call it the *base class validation accuracy*.

Novel class validation Another way of evaluating the quality of our representation learning stage is to use a disjoint set of classes from the base and novel classes. Concretely, we use a collection of examples $X^{val} := (x_1^{val}, \ldots, x_n^{val})$ with each $x_i^{val} \in \mathcal{X}$, and

corresponding labels $Y^{val} := (y_1^{val}, \dots, y_n^{val})$ with each $y_i^{val} \in C^{val}$, where $C^{val} := [c^{val}]$ is a set of *validation classes* disjoint from C and C'. Such collection might not exist in real life scenarios as we might want to use as many base classes as possible to learn a stronger representation of the data. In this case, we could use a cross-validation strategy. Usually, cross-validation consists in repeating the training process multiple times, each time retaining a different subset of the training samples to use as validation. In our few-shot learning context, the split between training and validation would be class-wise, that is, we would retain some training classes to use as validation classes. In the case of few-shot learning datasets, this collection is usually accessible, so cross-validation is not necessary.

By using this validation set we can simulate our few-shot testing setting and therefore evaluate the ability of our representation to generalize to novel classes. Because no classifier for the novel validation classes has been learned during the representation learning stage, we must choose and apply a stage 2 strategy to use this validation set. We choose to use the prototype classifier as a method to evaluate stage 1 performance. This choice is standard in few-shot learning, including our own research that we will develop in the rest of this manuscript. The few-shot classification accuracy of the model on this set is called the *novel class validation accuracy.*

Choice of validation metric We monitor the base class validation accuracy and novel class validation accuracy to choose a metric to use for the representation learning stage. For those experiments, we use the cosine classifier approach [QBL18; GK18] to optimize ResNet-18 using SGD with nesterov momentum 0.9 and weight decay $5e-4$ for 100 epochs. We apply this setup for 2 separate runs to alleviate the randomness of the initialization. In Figure 3.5, we show the evolution of both validation metrics during optimization. Both metrics are smoothed over multiple epochs for easier visualization. This experiment confirms that, as a general rule of thumb, the better the network has learned to classify over base classes, the better it can be used on novel classes in a few-shot scenario.

A closer observation reveals different phases of learning. During a first stage (up to about 55% base class validation accuracy), the optimization improves both the base class validation accuracy and the novel class validation accuracy. In this stage, the relationship between the two metrics is almost linear. Then during a second stage, the base class validation accuracy continue to improve while the novel class validation accuracy stagnates or even decreases in some cases. Finally we sometimes observe a last stage, where the base class accuracy also decreases.

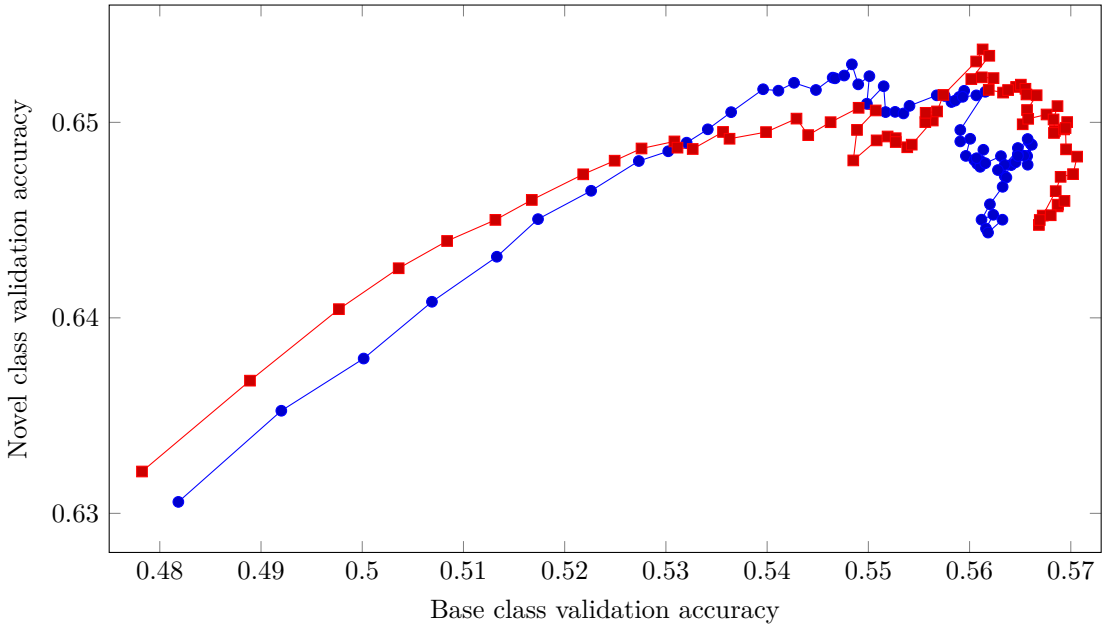


Figure 3.5 – Evolution of the base class validation accuracy and novel class validation accuracy during representation learning using cosine classifier, ResNet-18 as embedding network and constant learning rate. We show 2 runs of the same optimization.

This experiment confirms that our optimization using the base dataset is able to improve both the base class validation accuracy as well as the novel class validation accuracy. Both of those metrics agree at the beginning of the optimization process. After some epochs, the behaviour changes. The base class validation accuracy keeps increasing, which demonstrates that we are not overfitting on the training data. However, the drop in novel class validation accuracy shows that further optimization is not relevant for novel classes. Such behaviour can be interpreted as overfitting, not on the training data but on the training task (classification over the base classes).

In Table 3.1 we report all possible accuracy metrics for two optimized network. In both cases, the learning rate is kept fixed. For the first model we select the epoch with highest base class validation accuracy, while for the other we select the one with the highest novel class validation accuracy. We observe that despite having 4% lower base class validation accuracy, the second model performs better by about 2% for both 1-shot and 5-shot 5-way classification on the test set. This confirms that the correct choice of validation is the novel class accuracy, even for the representation learning stage. For this reason, we will solely use the novel class validation through this thesis.

Used validation	Base Validation	Novel validation		Test	
		1-shot	5-shot	1-shot	5-shot
Base class	59.34	49.91 ±0.46	65.42 ±0.39	47.76 ±0.44	63.04 ±0.38
Novel class	55.03	52.79 ±0.47	68.52 ±0.39	**49.63 ±0.43**	**65.26 ±0.37**

Table 3.1 – Comparison of the performance of the embedding network depending on the validation metric used for early stopping. The first model is selected from the best epoch regarding the base class validation accuracy, the second regarding novel class validation accuracy. For each we report the other metric result as well as few-shot test performance. All few-shot accuracy reported are computed on 5-way tasks of *mini*ImageNet with ResNet-18 as embedding network.

3.1.3 Selection of architecture and method

The quality of the representation model depends on the network architecture used to implement it as well as the method used to train it. In this section, we experiment with different combinations of those to choose the most relevant for our work.

We compare performances of the architecture of embedding networks most commonly used in the few-shot learning community. We perform our experiment using *mini*ImageNet as it is the most commonly used dataset. For representation learning we experiment with two standard methods: prototypical networks and cosine classifier. We compare few-shot classification accuracy using C64F, C128F, ResNet-12, and ResNet-18 as embedding networks. For C64F and C128F, we experiment with the original version where output embedding features are flattened, referred to respectively as C64F+Flatten and C128F+Flatten, as well as versions where the output embedding are pooled spatially using global pooling, referred to respectively as C64F+GAP and C128F+GAP. For inference, we use a prototype classifier.

We use stochastic gradient descent with nesterov momentum of 0.9 and weight decay of $5e-4$ in all experiments. For each experiment, we decrease the learning rate on the plateaus of the novel validation accuracy. Each reported result is the best out of 3 runs according to validation accuracy.

In Table 3.2 we report 5-way 1-shot, and 5-shot accuracy on *mini*ImagNet with those settings. Accuracies are averaged over 2000 few-shot tasks sampled from the test dataset with 15 queries per novel class for each. The best accuracy for 1-shot and 5-shot classification is put in bold. We observe that the choice of embedding network architecture and the choice of representation learning cannot be done entirely separately. Four layers convolutional networks with flattening of the feature maps perform better with

	1-shot		5-shot	
Network	PN	CC	PP	CC
C64F+Flatten	53.01 ±0.43	51.58 ±0.41	70.84 ±0.35	68.18 ±0.37
C128F+Flatten	53.57 ±0.43	52.45 ±0.41	71.42 ±0.36	69.53 ±0.35
C64F+GAP	52.10 ±0.45	51.95 ±0.43	69.08 ±0.36	68.69 ±0.36
C128F+GAP	52.58 ±0.45	52.70 ±0.42	69.71 ±0.37	69.99 ±0.36
ResNet-18	52.13 ±0.46	53.07 ±0.45	66.71 ±0.38	69.28 ±0.37
ResNet-12	60.09 ±0.47	**60.38 ±0.44**	75.02 ±0.35	**77.55 ±0.33**

Table 3.2 – Average 5-way accuracy on novel classes of *mini*ImageNet using different embedding networks and representation learning method. Novel class inference is performed using the prototype classifier method. PN: prototypical networks, CC: cosine classifier.

prototypical network than cosine classifier training (between 1% and 2% accuracy difference). Four layers convolutional networks with global pooling are performing about as well with both representation methods. Residual architectures perform better using the cosine classifier method with up to 2.6% accuracy increase on 5-way 5-shot classification with ResNet-18. ResNet-12 with cosine classifier is the best option out of all, showing the best accuracies in the two task settings, the second best being ResNet-12 with prototypical classifier training. For most of the experiments in this thesis, we will use residual network embdeddings with cosine classifier as baseline.

3.1.4 Number of shot

The term "few" in "few-shot learning" refers to having few training examples from the novel class set. As defined in section 2.2, we have n' labeled example for the novel class data. Here we ask ourselves how few should few-shot data be.

To do that we perform experiments on the mini-ImageNet dataset where the embedding network is ResNet-12 trained using the cosine classifier or prototypical network method on the base dataset. We compute the test accuracy by sampling 2000 few-shot tasks from the test set. Few-shot tasks are 5-way k-shot (we sample k support examples per novel class). We make k vary from 1 to 20. Results are reported in Figure 3.6. We observe that the addition of extra support examples makes a very large difference when the initial number of shot is very low. For instance, there is a 6.8% increase in accuracy when going from 1-shot to 2-shot using the cosine classifier method. The impact is lower with higher numbers of shot, with improvements of maximum 0.5% accuracy per extra shot for 10-shot and higher. This behaviour is easily explainable since the necessity of

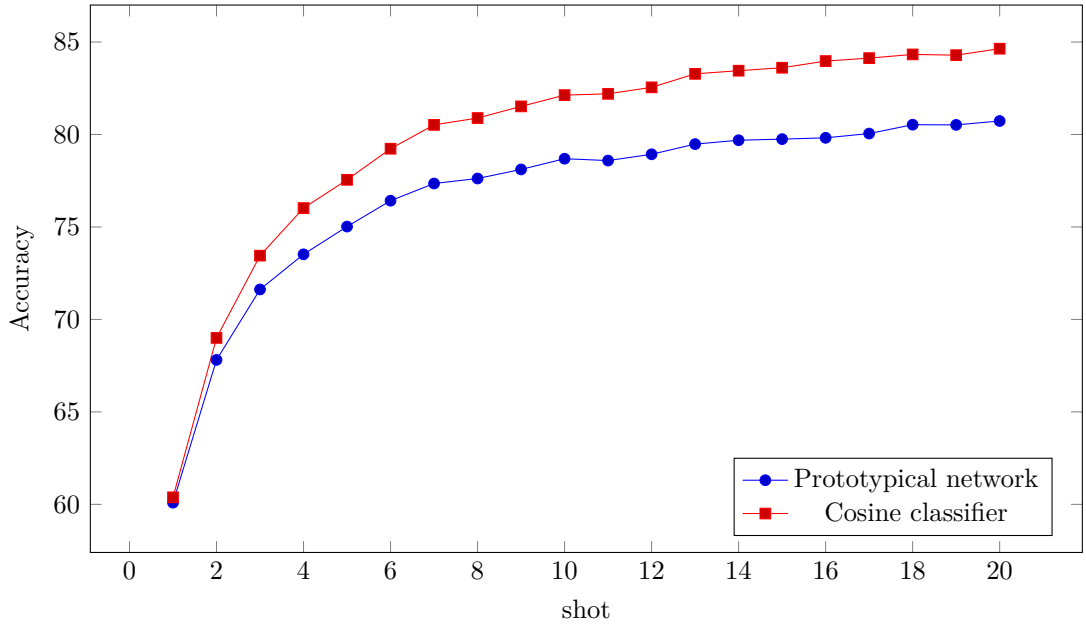


Figure 3.6 – *mini*ImageNet few-shot 5-way accuracy with varying number of shot. The embedding network is ResNet-12.

gathering more information about the novel classes is more important if such information is lacking which is the case with very few shots. In order to stay true to the idea of few-shot learning, we choose to focus on this very few-shot setting, that is to say, evaluating models based on how they perform 1-shot, 5-shot or at maximum 10-shot classification. This choice is consistent with what can be found in the related literature.

3.2 Dense Classification

3.2.1 Method

As discussed in section 2.2, the *embedding network* $\phi_\theta : \mathcal{X} \to \mathbb{R}^{r \times d}$ maps the input to an embedding that is a tensor. There are two common ways of handling this high-dimensional representation, as illustrated in Figure 3.7.

The first is to apply one or more fully connected layers. This can be seen as *flattening* the activation into a long vector and multiplying with a weight vector of the same length per class; alternatively, the weight parameter is a tensor of the same dimension as the embedding. This representation is discriminative, but not invariant. This was standard in many early few-shot learning works [Vin+16; SSZ17; GK18], generally using small convolutional networks like C64F or C128F for image representation, then classifying

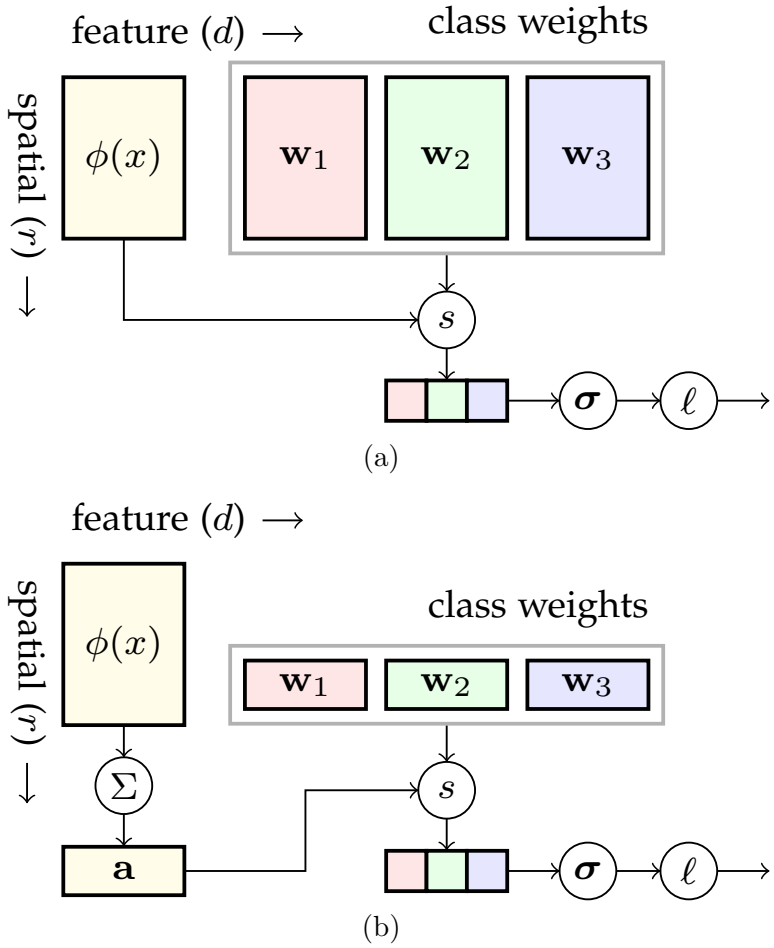


Figure 3.7 – *Flattening and pooling.* Horizontal (vertical) axis represents feature (spatial) dimensions. Tensors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ represent class weights, and $\phi(\mathbf{x})$ the embedding of example $\mathbf{x}$. An embedding is compared to class weights by similarity (s) and then softmax ($\boldsymbol{\sigma}$) and cross-entropy (ℓ) follow. (a) *Flattening* is equivalent to class weights having the same $r \times d$ shape as $\phi(\mathbf{x})$. (b) *Global pooling.* Embedding $\phi(\mathbf{x})$ is pooled (Σ) into vector $\mathbf{a} \in \mathbb{R}^d$ before being compared to class weights, which are in $\mathbb{R}^d$ too.

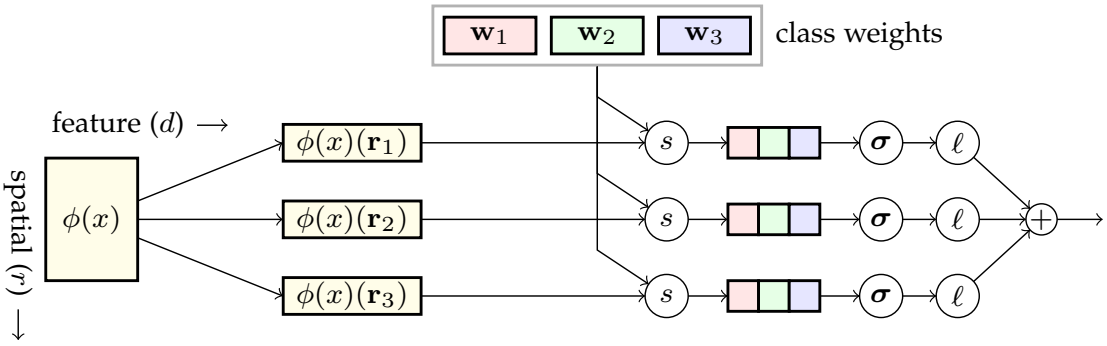


Figure 3.8 – *Dense classification.* Notation is the same as in Figure 3.7. The embedding $\mathbf{a} := \phi(\mathbf{x}) \in \mathbb{R}^{r \times d}$ is seen as a collection of vectors $(\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(r)})$ in $\mathbb{R}^d$ (here $r = 3$) with each being a vector in $\mathbb{R}^d$ and representing a region of the input image. Each vector is compared independently to the same class weights and the losses are added, encouraging all regions to be correctly classified.

using a single or multi-layer neural network.

The second way is to apply *global pooling* and reduce the embedding into a smaller vector of length d. This reduces dimensionality significantly, so it only makes sense if d is large enough. It is an invariant representation, but less discriminative. This method is used in more recent few-shot learning works that used ResNet architecture as embedding networks that have high dimensional outputs [Mis+18; GK18; OLR18].

We propose a different approach that we call *dense classification* and is illustrated in Figure 3.8. We view the embedding $\phi_\theta(\mathbf{x})$ as a collection of vectors $[\phi^{(k)}(\mathbf{x})]_{k=1}^r$, where $\phi^{(k)}(\mathbf{x}) \in \mathbb{R}^d$ for $k \in [r]$[1]. For a 2d image input and a convolutional network, $\phi_\theta(\mathbf{x})$ consists of the activations of the last convolutional layer, that is a tensor in $\mathbb{R}^{w \times h \times d}$ where $r = w \times h$ is its spatial resolution. Then, $\phi^{(k)}(\mathbf{x})$ is an embedding in $\mathbb{R}^d$ that represents a single spatial location k on the tensor.

When learning from the training data (X, Y) over base classes C (stage 1), we adopt the simple approach of training a *parametric linear classifier* on top of the embedding function ϕ_θ, like [QBL18] and the initial training of [GK18]. The main difference in our case is that the weight parameters do *not* have the same dimensions as $\phi_\theta(\mathbf{x})$; they are rather vectors in $\mathbb{R}^d$ and they are *shared* over all spatial locations. More formally, let $\mathbf{w}_j \in \mathbb{R}^d$ be the weight parameter of class j for $j \in C$. Then, similarly to (2.15), the

[1]Given tensor $\mathbf{a} \in \mathbb{R}^{m \times n}$, denote by $\mathbf{a}^{(k)}$ the k-th n-dimensional slice along the first group of dimensions for $k \in [m]$.

classifier mapping $f_{\theta,W} : \mathcal{X} \to \mathbb{R}^{r\times c}$ is defined by

$$f_{\theta,W}(\mathbf{x}) := \left[\boldsymbol{\sigma}\left([s_\tau(\phi_\theta^{(k)}(\mathbf{x}), \mathbf{w}_j)]_{j=1}^c\right)\right]_{k=1}^r \tag{3.1}$$

for $\mathbf{x} \in \mathcal{X}$, where $W := (\mathbf{w}_1, \ldots, \mathbf{w}_c)$ is the collection of class weights and s_τ is the *scaled cosine similarity* defined by (2.16), with τ being a learnable parameter as in [QBL18; GK18][2]. Here $f_{\theta,W}(\mathbf{x})$ is a $r \times c$ tensor: index k ranges over spatial resolution $[r]$ and j over classes $[c]$.

This operation is a 1×1 *convolution followed by depth-wise softmax.* Then, $f_{\theta,W}^{(k)}(\mathbf{x})$ at spatial location k is a vector in $\mathbb{R}^c$ representing confidence over the c classes. On the other hand, $f_{\theta,W}^{(:,j)}(\mathbf{x})$ is a vector in $\mathbb{R}^r$ representing confidence of class j for $j \in [c]$ as a function of spatial location.[3] For a 2d image input, $f_{\theta,W}^{(:,j)}(\mathbf{x})$ is like a *class activation map* (CAM) [Zho+16] for class j, that is a 2d map roughly localizing the response to class j, but differs in that softmax suppresses all but the strongest responses at each location.

Given the definition (3.1) of $f_{\theta,W}$, training amounts to minimizing over θ, W the *cost function*

$$J(X, Y; \theta, W) := \sum_{i=1}^n \sum_{k=1}^r \ell(f_{\theta,W}^{(k)}(\mathbf{x}_i), y_i), \tag{3.2}$$

where ℓ is cross-entropy (2.2). The loss function applies to all spatial locations and therefore the classifier is encouraged to make correct predictions everywhere.

3.2.2 Discussion

Similarities can be found with *semantic segmentation* [LSD15; NHH15] , where given per-pixel labels, the loss function applies per pixel and the network learns to make localized predictions on upsampled feature maps rather than just classify. In our case there is just one image-level label. In the case of the original version of *mini*ImageNet, using ResNet-12 as embedding network, the features maps have low resolution (5×5). Moreover relevant object of *mini*ImageNet images take a large portion of the image. In this case, the receptive field of a particular location is large enough to assume we can assign it the image label. In case of larger images, this assumption might not hold.

[2] *Temperature scaling* is frequently encountered in various formats in several works to enable soft-labeling [HVD15] or to improve cosine similarity in the final layer [Wan+17a; OLR18; GK18; QBL18; HHS18].

[3] Given tensor $\mathbf{a} \in \mathbb{R}^{m\times n}$, denote by $\mathbf{a}^{(:,j)}$ the j-th m-dimensional slice along the second group of dimensions for $j \in [n]$.

Figure 3.9 – Examples overlaid with correct *class activation maps* [Zho+16] (red is high activation for ground truth) on ResNet-12 trained with global average pooling or dense classification (*cf.* (3.1)). From top to bottom: base classes, classified correctly by both (walker hound, tile roof); novel classes, classified correctly by both (king crab, ant); novel classes, dense classification is better (ferret, electric guitar); novel classes, pooling is better (mixing bowl, ant). In all cases, dense classification results in smoother activation maps that are more aligned with objects.

In this case, we can introduce a local average pooling operation before applying dense classification to expand the receptive field of each location.

Dense classification improves the spatial distribution of class activations, as shown in Figure 3.9. By encouraging all spatial locations to be classified correctly, we are encouraging the embedding network to identify all parts of the object of interest rather than just the most discriminative details. Since each location on a feature map corresponds to a region in the image where only part of the object may be visible, our model behaves like *implicit data augmentation* of exhaustive shifts and crops over a dense grid with a single forward pass of each example in the network.

3.2.3 Inference on novel classes

In Figure 3.2.1 we define a new way of handling spatial information during while training an embedding network with a classifier for classification. In this case the classifier being a one layer cosine classifier with learned scaling. We propose to use this new training method for stage 1 of our few-shot learning model. At inference, we are again confronted with a choice on how to handle spatial information of features maps from the support and query examples. We adopt the prototype classifier model for inference. With this choice, we have found that it is working best to perform global pooling of the support examples before computing class prototypes $P := (\mathbf{p}_1, \dots, \mathbf{p}_{c'})$ by (2.12). Given a query $\mathbf{x} \in \mathcal{X}$, the standard prediction is then to assign it to the nearest prototype

$$\arg\max_{j \in C'} s(\phi_{\theta,\theta'}(\mathbf{x}), \mathbf{p}_j), \tag{3.3}$$

where s is cosine similarity [SSZ17]. Alternatively, we can use *dense classification* on queries, that is to say soft-assigning independently the embedding $\phi^{(k)}_{\theta,\theta'}(\mathbf{x})$ of each spatial location, then average over all locations $k \in [r]$ according to

$$f_{\theta,\theta'}[P](\mathbf{x}) := \frac{1}{r} \sum_{k=1}^{r} \boldsymbol{\sigma}\left([s_\tau(\phi^{(k)}_{\theta,\theta'}(\mathbf{x}), \mathbf{p}_j)]_{j=1}^{c'} \right), \tag{3.4}$$

where s_τ is the scaled cosine similarity (2.16), and finally classify to $\arg\max_{j \in C'} f^j_{\theta,\theta'}[P](\mathbf{x})$.

3.3 Experiments

We evaluate our *dense classification* method extensively on the *mini*ImageNet and FC100 datasets. We describe the experimental setup and report the results below.

3.3.1 Experimental setup

Networks In most experiments, we use a ResNet-12 network [OLR18] as our embedding network. Because our dense classification method might affect the results of the network study presented in subsection 3.1.3, we also test dense classification on a lighter network C128F [GK18].

Datasets We test our method on the *mini*ImageNet dataset, as well as CIFAR100.

Network	Pooling	1-shot	5-shot	10-shot
C128F	GAP	54.28 ±0.18	71.60 ±0.13	76.92 ±0.12
C128F	DC	49.84 ±0.18	69.64 ±0.15	74.61 ±0.13
ResNet-12	GAP	58.61 ±0.18	76.40 ±0.13	80.76 ±0.11
ResNet-12	DC	61.26 ±0.20	79.01 ±0.13	83.04 ±0.12

Table 3.3 – *Average 5-way accuracy on novel classes of* mini*ImageNet, stage 1 only.* Pooling refers to stage 1 training. GAP: global average pooling; DC: dense classification. At testing, we use global max-pooling on queries for models trained with dense classification, and global average pooling otherwise.

Evaluation protocol The training set X comprises images of the base classes C. To generate the support set X' of a few-shot task on novel classes, we randomly sample C' classes from the validation or test set and from each class we sample k images. We report the average accuracy and the corresponding 95% confidence intervals over 10,000 few-shot tasks with 30 queries per class. Using the same task sampling, we also consider few-shot tasks involving base classes C, following the benchmark of [GK18]. We sample a set of extra images from the base classes to form a test set for this evaluation, which is performed in two ways: independently of the novel classes C' and jointly on the union $C \cup C'$. In the latter case, the base prototypes learned at stage 1 are concatenated with novel prototypes [GK18].

Implementation details In stage 1, we train the embedding network for 8,000 (12,500) iterations with mini-batch size 200 (512) on *mini*ImageNet (FC100). On *mini*ImageNet, we use stochastic gradient descent with Nesterov momentum. On FC100, we rather use Adam optimizer [KB14]. We initialize the scale parameter at $\tau = 10$ (100) on *mini*ImageNet (FC100).

3.3.2 Results

Networks In Table 3.3 we compare ResNet-12 to C128F, with and without dense classification. We observe that dense classification improves the classification accuracy on novel classes for ResNet-12, but it is detrimental for the small network. C128F is only 4 layers deep and the receptive field at the last layer is significantly smaller than the one of ResNet-12, which is 12 layers deep. It is thus likely that units from the last feature map correspond to non-object areas in the image. Regardless of the choice of using dense classification or not, ResNet-12 has a large performance gap over C128F.

For the following experiments, we use exclusively ResNet-12 as our embedding network.

Stage 1 training		Support/query pooling at testing			
	Support →	GMP		GAP	
	Queries →	GMP	DC	GAP	DC
GAP	Base classes	63.55 ±0.20	77.17 ±0.11	79.37 ±0.09	77.15 ±0.11
	Novel classes	72.25 ±0.13	70.71 ±0.14	76.40 ±0.13	73.28 ±0.14
	Both classes	37.74 ±0.07	38.65 ±0.05	56.25 ±0.10	54.80 ±0.09
DC	Base classes	79.28 ±0.10	**80.67 ±0.10**	**80.61 ±0.10**	**80.70 ±0.10**
	Novel classes	**79.01 ±0.13**	77.93 ±0.13	78.55 ±0.13	**78.95 ±0.13**
	Both classes	42.45 ±0.07	57.98 ±0.10	67.53 ±0.10	**67.78 ±0.10**

Table 3.4 – *Average 5-way 5-shot accuracy on base, novel and both classes of* mini*ImageNet with ResNet-12, stage 1 only.* GMP: global max-pooling; GAP: global average pooling; DC: dense classification. Bold: accuracies in the confidence interval of the best one.

Dense classification In Table 3.4 we evaluate 5-way 5-shot classification on *mini*-ImageNet with global average pooling and dense classification at stage 1 training, while exploring different pooling strategies at inference. We also tried using global max-pooling at stage 1 training and got similar results as with global average pooling. Dense classification in stage 1 training outperforms global average pooling in all cases by a large margin. It also improves the ability of the network to integrate new classes without forgetting the base ones. Using dense classification at testing as well, the accuracy on both classes is 67.78%, outperforming the best result of 59.35% reported by [GK18]. At testing, dense classification of the queries with global average pooling of the support examples is the best overall choice. One exception is global max-pooling on both the support and query examples, which gives the highest accuracy for new classes but the difference is insignificant.

In order to further investigate the impact of using dense classification in stage 1, we computed 5-way accuracy with the number of shots ranging from 1 to 20 both with our dense classification method and global pooling. We report in Figure 3.10 the improvements in accuracy for all those task settings. We observe that the largest accuracy improvement (3.8%) is for the 1-shot setting. The relative gain decreases with the number of shot. Nevertheless, even in the worse case we observe around 1.5% accuracy improvement brought by dense classification.

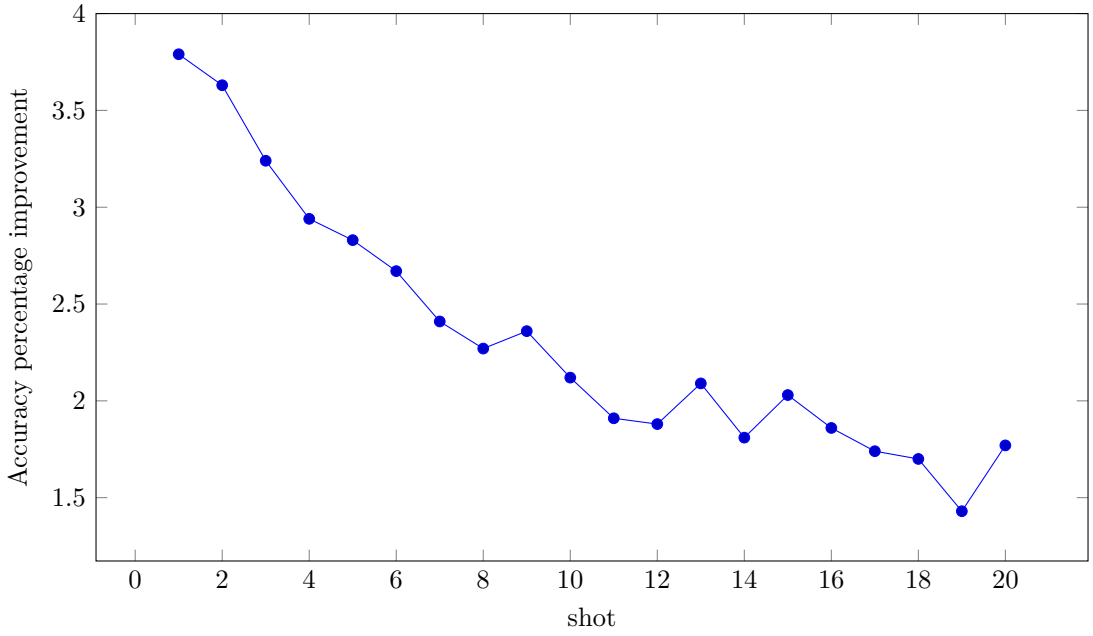


Figure 3.10 – *mini*ImageNet few-shot 5-way accuracy improvement when using dense classification compared to global average pooling training, with varying number of shot. ResNet-12 is used as embedding network.

Comparison with the state-of-the-art In Table 3.5 we compare our best dense classification solution with previous few-shot learning methods on 5-way classification on *mini*ImageNet. All methods included are using the same ResNet-12 embedding network for fairness of comparison. We also include results from our baseline which is the embedding network trained with global average pooling in stage 1. Our solution outperforms by at least 2% accuracy other methods on 1, 5 and 10-shot classification. Note that prototypical network on ResNet-12 [OLR18] is already giving very competitive performance. TADAM [OLR18] builds on top of this baseline to achieve the previous state of the art. In this work we rather use a cosine classifier in stage 1. This setting is our baseline GAP and is already giving similar performance to TADAM [OLR18]. Dense classification is able to improve on this baseline. Similar conclusions can be drawn from Table 3.6, showing corresponding results on FC100. When comparing to prototypical network [OLR18] and TADAM [OLR18], Our model outperforms TADAM here too, although by a smaller margin. The lower resolution of CIFAR-100, may be the cause of lower gain from using dense classification since the cost function (3.2) applies to fewer spatial locations.

Method	1-shot	5-shot	10-shot
GAP	58.61 ± 0.18	76.40 ± 0.13	80.76 ± 0.11
DC (ours)	**62.53 ± 0.19**	**78.95 ± 0.13**	**82.66 ± 0.11**
MAML [FAL17]	48.70 ± 1.8	63.10 ± 0.9	-
PN [SSZ17]	49.42 ± 0.78	68.20 ± 0.66	-
Gidaris et al. [GK18]	55.45 ± 0.7	73.00 ± 0.6	-
PN [OLR18]	56.50 ± 0.4	74.20 ± 0.2	78.60 ± 0.4
TADAM [OLR18]	58.50	76.70	80.80

Table 3.5 – *Average 5-way accuracy on novel classes of* mini*ImageNet*. The top part is our solutions and baselines, all on ResNet-12. GAP: global average pooling (stage 1); DC: dense classification (stage 1). At testing, we use GAP on support examples and GAP or DC on queries, depending on the choice of stage 1. The bottom part results are as reported in the literature. PN: Prototypical Network [SSZ17]. MAML [FAL17] and PN [SSZ17] use four-layer networks; while PN [OLR18] and TADAM [OLR18] use the same ResNet-12 as us. Gidaris et al. [GK18] use a Residual network of comparable complexity to ours.

Method	1-shot	5-shot	10-shot
GAP	41.02 ± 0.17	56.63 ± 0.16	61.65 ±0.15
DC (ours)	**42.04 ± 0.17**	**57.05 ± 0.16**	**61.91 ± 0.16**
PN [OLR18]	37.80 ± 0.40	53.30 ± 0.50	58.70 ± 0.40
TADAM [OLR18]	40.10 ± 0.40	56.10 ± 0.40	61.60 ± 0.50

Table 3.6 – *Average 5-way accuracy on novel classes of FC100 with ResNet-12*. The top part is our solutions and baselines. GAP: global average pooling (stage 1); DC: dense classification (stage 1). At testing, we use GAP on support examples and GAP or DC on queries, depending on the choice of stage 1. The bottom part results are as reported in the literature. All experiments use the same ResNet-12.

3.4 Conclusion

We have seen that the way representation learning is approached in stage 1 is crucial for few-shot performance even though the classes on which the latter is evaluated are different from the ones used during training. We have experimented with two simple methods used in few-shot learning works: prototypical networks and cosine classifier. We have identified that the cosine classifier method is the most promising, and that deep architectures such as ResNet-12 are the most effective for modern few-shot learning datasets. This setting, with hyperparameters chosen using a validation set composed of

images from a disjoint set of classes from the base and novel classes, allow impressive results already.

We also bring a contribution to few-shot learning by building on this simple process. We investigate for the first time in few-shot learning the activation maps and devise a new way of handling spatial information by a dense classification loss that is applied to each spatial location independently, significantly improving the spatial distribution of the activation and performance on novel classes or when presented with a mix of base and novel classes.

Chapter 4

Adaptation of the representation to the few-shot task

Contents

The success of deep learning methods for computer vision tasks is due to the capacity of training a powerful representation of images that groups images with similar content together. This representation is obtained by minimizing a *loss function* to optimize the

parameters of the network. Many different losses can be used to get to such semantic representation. *Metric learning* losses [HA15; Oh +16; Soh16; Wan+17c] directly enforce mapping similar images close together in some feature space and dissimilar images far away from each other. *Classification losses* such as the cross-entropy loss enforces good classification of the images. Again, classification implies learning to represent images in a feature space, where images from the same classes are grouped together. The learned representation is especially powerful since it is specifically designed to work well on the training data. In a traditional classification problem, the training dataset is a large set of examples (e.g. thousands) from the classes that we want to classify into.

However, in few-shot learning, examples from the novel classes are limited to a few ones, tens at maximum. This small amount of data is not enough to train deep network as already discussed in the previous chapter. We have seen how we can use a larger set of images, the base class dataset, to train the representation instead. This representation alone with a simple classifier for the novel classes, a prototype classifier for instance, is enough to have impressive classification results on few-shot tasks. However, this method ignores the specificity of the novel classes in the representation, assuming that the representation learned on base classes is enough.

In this chapter, we explore ways to adapt the representation of the images to the novel data using the few-shot examples only to make it specific to the few-shot task. Naturally, the objective is not to completely deviate from the representation learned on the base class dataset, because we would run the risk of overfitting.

Instead, we propose two learning-based methods to slightly modify the representation. The first one is called *implanting* and has been published in [Lif+19]. It consists in adding a limited number of new parameters, called implants, to the trained embedding network. Only the implants alone are learned on the few-shot data. The outputs of the implants are used as new dimensions in the vector representation of the images. Learning a limited number of parameters allows multiple iterations of learning without overfitting.

The second method is a simple fine-tuning of the embedding network with the few-shot data and is part of [LAP20a]. With many iterations, we end up overfitting because of the lack of data. To prevent this overfitting, we limit the training to a few iterations, allowing on average to improve the few-shot accuracy.

4.1 Implanting

From the learning on the training data (X, Y) of base classes C (representation learning stage) we only keep the embedding network ϕ_θ and we discard the classification layer. The assumption is that features learned on base classes are generic enough to be used for other classes, at least for the bottom layers [Yos+14]. However, given a new few-shot task on novel classes C', we argue that we can take advantage of the support data (X', Y') to find new features that are discriminative for the task at hand, at least in the top layers. In this section, we propose a novel solution to perform this adaptation through learning on the novel data without overfitting. Overfitting happens because the number of parameters to learn is too large for the available data. Therefore, our solution is to only learn a small number of parameters added to the embedding network. Original parameters are frozen so that they are not squashed during the new training phase.

4.1.1 Related works

Network adaptation is common when learning a new task or new domain. One solution is to learn to *mask* part of the network, keeping useful neurons and re-training/fine-tuning the remaining neurons on the new-task [MDL18; ML18]. Rusu *et al.* [Rus+16] rather *widen* the network by adding new neurons in parallel to the old ones at every layer. New neurons receive data from all hidden states, while previously generated weights are frozen when training for the new task. Our neural implants are related to [Rus+16] as we add new neurons in parallel and freeze the old ones. Unlike [Rus+16], we focus on low-data regimes, keeping the number of new implanted neurons small to diminish overfitting risks and train faster, and adding them only at top layers, taking advantage of generic visual features from bottom layers. Parallel to our work, Sun et al. [Sun+19] proposed a related solution where task-dependent scaling and shifting parameters are learned for each layer.

4.1.2 Architecture

We begin with the embedding network ϕ_θ, which we call *base network*. We widen this network by adding new convolution kernels in a number of its top convolutional layers. We call these new neurons *implants*. While learning the implants, we keep the base network parameters frozen, which preserves the representation of the base classes.

Let $\mathbf{a}_l$ denote the output activation of the convolutional layer l in the base network. The implant for this layer, if it exists, is a distinct convolutional layer with output

activation $\mathbf{a}'_l$. Then the input of an implant at the next layer $l+1$ is the depth-wise concatenation $[\mathbf{a}_l, \mathbf{a}'_l]$ if $\mathbf{a}'_l$ exists, and just $\mathbf{a}_l$ otherwise. If θ'_l are the parameters of the l-th implant, then we denote by $\theta' := (\theta'_{l_0}, \ldots, \theta'_L)$ the set of all new parameters, where l_0 is the first layer with an implant and L the network depth. The *widened* embedding network is denoted by $\phi_{\theta,\theta'}$.

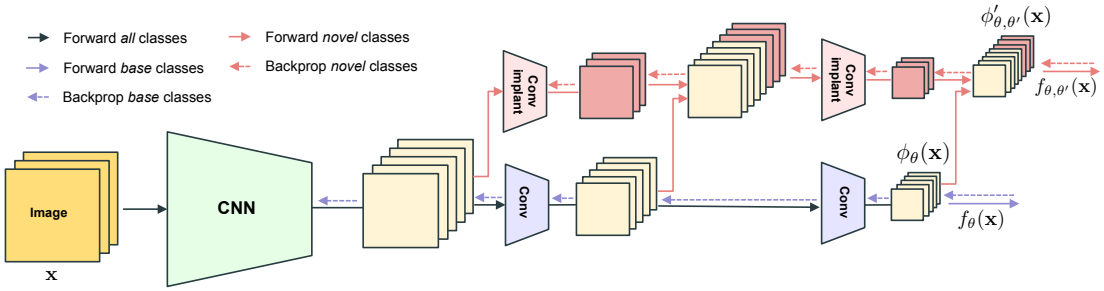


Figure 4.1 – *Neural implants for CNNs.* The implants are convolutional filters operating in a new processing stream parallel to the base network. The input of an implant is the depth-wise concatenation of hidden states from both streams. When training neural implants, previously trained parameters are frozen. Purple and black arrows correspond to the representation learning stage flows; red and black to adaptation.

As illustrated in Figure 4.1, we are creating a new stream of data in parallel to the base network. The implant stream is connected to the base stream at multiple top layers and leverages the previously learned features by learning additional connections for the new tasks. For simplicity sake, in this example, the original embedding network is a simple feed forward convolutional neural network. In our experiments, we use residual network architectures as described in section 3.1. In this case, each convolutional block inside of a residual block is widened as show in Figure 4.2, resulting in a skip connection for the implant stream as well.

4.1.3 Training

When a new task is given, we want to learn the implant parameters on the support data (X', Y') over novel classes C'. Here we use an approach similar to prototypical networks [SSZ17] in the sense that we generate a number of fictitious *subtasks* of the new task, the main difference being that we are now working on the novel classes.

We choose the simple approach of using each one of the given examples alone as a query in one subtask while all the rest are used as support examples. This involves no sampling and the process is deterministic. Because only one example is missing from the true support examples, each subtask approximates the true task very well.

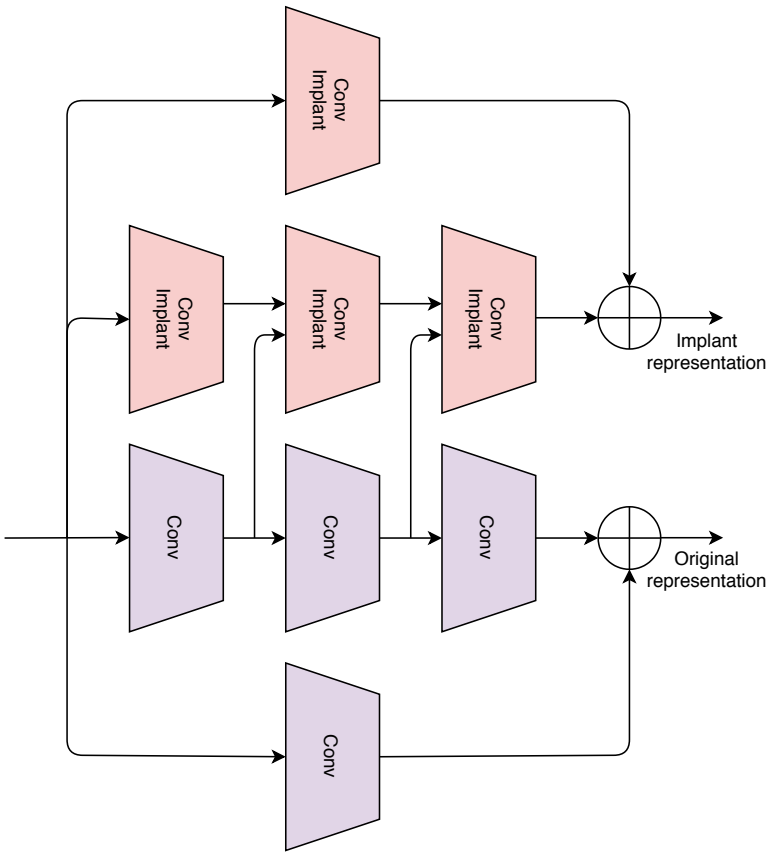


Figure 4.2 – *Neural implants for a residual block.* Implants replicate the residual structure of the block, while utilizing the original data stream for each convolutional block.

In parallel to our work, Wertheimer and Hariharan [WH19] introduced a similar process called *batch folding*. They propose an alternative to the prototypical network base learning stage where all subtasks possible within a batch are considered. The difference is that we do this at adaptation stage.

In particular, for each $i \in N' := [n']$, we define a *query set* $Q_i := \{i\}$ and a *support set* $S_i := N' \setminus Q_i$. We compute class prototypes P_i on index set S_i according to (2.12), where we replace ϕ_θ by $\phi_{\theta,\theta'}$ and θ' are the implanted parameters. We define the *widened* network function $f_{\theta,\theta'}[P_i]$ on these prototypes by (2.13) with a similar replacement. We then freeze the base network parameters θ and train the implants θ' by minimizing a cost function like (2.17). Similarly to (2.17) and taking all subtasks into account, the

overall cost function we are minimizing over θ' is given by

$$J(X', Y'; \theta, \theta') := \sum_{i=1}^{n'} \ell(f_{\theta,\theta'}[P_i](\mathbf{x}'_i), y'_i), \tag{4.1}$$

where ℓ is cross-entropy (2.2).

In (4.1), activations are assumed flattened or globally pooled. Alternatively, we can densely classify them and apply the loss function to all spatial locations independently. Combining with (3.2), the cost function in this case is

$$J(X', Y'; \theta, \theta') := \sum_{i=1}^{n'} \sum_{k=1}^{r} \ell(f^{(k)}_{\theta,\theta'}[P_i](\mathbf{x}'_i), y'_i). \tag{4.2}$$

Prototypes in (4.1) or (4.2) are recomputed at each iteration based on the current version of implants. Note that this training setup does not apply to the 1-shot scenario as it requires at least two *support* examples per class.

4.1.4 Inference on novel classes

Inference is the same whether the embedding network has been implanted or not. Here we adopt the prototypical network model too. What we have found to work best is to perform global pooling of the embeddings of the support examples and compute class prototypes $P := (\mathbf{p}_1, \ldots, \mathbf{p}_{C'})$ by (2.12). Given a query $\mathbf{x} \in \mathcal{X}$, the standard prediction is then to assign it to the nearest prototype

$$\arg\max_{j \in C'} s(\phi_{\theta,\theta'}(\mathbf{x}), \mathbf{p}_j), \tag{4.3}$$

where s is cosine similarity [SSZ17]. Alternatively, we can densely classify the embedding $\phi_{\theta,\theta'}(\mathbf{x})$, soft-assigning independently the embedding $\phi^{(k)}_{\theta,\theta'}(\mathbf{x})$ of each spatial location, then average over all locations $k \in [r]$ according to

$$f_{\theta,\theta'}[P](\mathbf{x}) := \frac{1}{r} \sum_{k=1}^{r} \boldsymbol{\sigma}\left([s_\tau(\phi^{(k)}_{\theta,\theta'}(\mathbf{x}), \mathbf{p}_j)]_{j=1}^{C'} \right), \tag{4.4}$$

where s_τ is the scaled cosine similarity (2.16), and finally classify to $\arg\max_{j \in C'} f^j_{\theta,\theta'}[P](\mathbf{x})$.

4.2 Implanting experiments

4.2.1 Experimental setup

This set of experiments is in continuity with the ones described in section 3.3. Therefore the setup is similar.

Networks We use a ResNet-12 network [OLR18] as our embedding network. We have seen in section 3.3 that it performs better than the lighter C128F [GK18] network.

Datasets We test our method on the *mini*ImageNet dataset, as well as CIFAR100.

Evaluation protocol The training set X comprises images of the base classes C. To generate the support set X' of a few-shot task on novel classes, we randomly examples C' classes from the validation or test set and from each class we sample k images. We report the *average accuracy* and the corresponding 95% *confidence interval* over 10,000 few-shot tasks with 30 queries per class.

Implementation details In the representation learning stage, we train the embedding network for 8,000 (12,500) iterations with mini-batch size 200 (512) on *mini*ImageNet (FC100). On *mini*ImageNet, we use stochastic gradient descent with Nesterov momentum. On FC100, we rather use Adam optimizer [KB14]. We initialize the scale parameter at $\tau = 10$ (100) on *mini*ImageNet (FC100). For a given few-shot task in stage 2, the implants are learned over 50 epochs with AdamW optimizer [LH19] and scale fixed at $\tau = 10$.

4.2.2 Results

Implanting In stage 2, we add implants of 16 channels to all convolutional layers of the last residual block of our embedding network pre-trained in stage 1 on the base classes with dense classification. The implants are trained on the few examples of the novel classes and then used as an integral part of the widened embedding network $\phi_{\theta,\theta'}$ at testing. In Table 4.1, we evaluate different pooling strategies for support examples and queries in stage 2. Average pooling on both is the best choice, which we keep in the following.

Stage 2 training		Query pooling at testing		
Support	Queries	GAP	GMP	DC
GMP	GMP	79.03 ± 0.19	78.92 ± 0.19	79.04 ± 0.19
GMP	DC	79.06 ± 0.19	79.37 ± 0.18	79.15 ± 0.19
GAP	GAP	79.62 ± 0.19	74.57 ± 0.22	**79.77 ± 0.19**
GAP	DC	79.56 ± 0.19	74.58 ± 0.22	79.52 ± 0.19

Table 4.1 – *Average 5-way 5-shot accuracy on novel classes of* mini*ImageNet with ResNet-12 and implanting in stage 2.* At testing, we use GAP for support examples. GMP: global max-pooling; GAP: global average pooling; DC: dense classification.

Ablation study In the top part of Table 4.2 we compare our best solutions with a number of baselines on 5-way *mini*ImageNet classification. One baseline is the embedding network trained with global average pooling training in stage 1. As seen in the previous chapter, dense classification is the preferred method. In stage 2, the implants are able to further improve on the results of dense classification. To illustrate that our gain does not come just from having more parameters and greater feature dimensionality, another baseline is to compare it to widening the last residual block of the network by 16 channels in stage 1. It turns out that such widening does not bring any improvement on novel classes. Similar conclusions can be drawn from the top part of Table 4.3, showing corresponding results on FC100. The difference between different solutions is less striking here. This may be attributed to the lower resolution of CIFAR-100, allowing for less gain from either dense classification or implanting, since there may be less features to learn.

Comparison with the state-of-the-art. In the bottom part of table 4.2 we compare our model with previous few-shot learning methods on the same 5-way *mini*ImageNet classification. Implanting improving the results of dense classification alone, we obtain even better results than the ones displayed in the previous chapter. Our best results are at least 3% above TADAM [OLR18] in all settings. Finally, in the bottom part of Table 3.6 we compare our model on 5-way FC100 classification against prototypical network [OLR18] and TADAM [OLR18]. There too, we observe additional improvements compared to dense classification, resulting in higher performance than previous models.

Method	1-shot	5-shot	10-shot
GAP	58.61 ± 0.18	76.40 ± 0.13	80.76 ± 0.11
DC (ours)	**62.53 ± 0.19**	78.95 ± 0.13	82.66 ± 0.11
DC + WIDE	61.73 ± 0.19	78.25 ± 0.14	82.03 ± 0.12
DC + IMP (ours)	-	**79.77 ± 0.19**	**83.83 ± 0.16**
MAML [FAL17]	48.70 ± 1.8	63.10 ± 0.9	-
PN [SSZ17]	49.42 ± 0.78	68.20 ± 0.66	-
Gidaris et al. [GK18]	55.45 ± 0.7	73.00 ± 0.6	-
PN [OLR18]	56.50 ± 0.4	74.20 ± 0.2	78.60 ± 0.4
TADAM [OLR18]	58.50	76.70	80.80

Table 4.2 – *Average 5-way accuracy on novel classes of* mini*ImageNet*. The top part is our solutions and baselines, all on ResNet-12. GAP: global average pooling (stage 1); DC: dense classification (stage 1); WIDE: last residual block widened by 16 channels (stage 1); IMP: implanting (stage 2). In stage 2, we use GAP on both support and queries. At testing, we use GAP on support examples and GAP or DC on queries, depending on the choice of stage 1. The bottom part results are as reported in the literature. PN: Prototypical Network [SSZ17]. MAML [FAL17] and PN [SSZ17] use four-layer networks; while PN [OLR18] and TADAM [OLR18] use the same ResNet-12 as us. Gidaris et al. [GK18] use a Residual network of comparable complexity to ours.

4.3 Few-steps adaptation

Instead of limiting the number of parameters to learn as done with the implanting method, it is possible to attempt to fine-tune some layers or even the entirety of the embedding network. As stated before, the risk here is to run into overfitting. Overfitting is typically prevented using regularization methods such as weight decay [KH92] or dropout [Sri+14]. We experimented with regularization methods for few-shot adaptation unsuccessfully.

In this section, we propose to adapt the embedding network for a few steps of training only to limit overfitting. We study the impact of the number of learning steps at adaptation on the final classification accuracy.

4.3.1 Related works

Some influential few-shot learning works also propose to adapt the embedding network without extra regularization methods. Specifically, Qi et al. [QBL18] fine-tune the entire embedding network using the few-shot data in addition to all base data. They oversample the few-shot data to balance the class distribution during adaptation. In this way, they avoid the need for strong regularization. However, this method requires to have access

Method	1-shot	5-shot	10-shot
GAP	41.02 ± 0.17	56.63 ± 0.16	61.65 ±0.15
DC (ours)	**42.04 ± 0.17**	57.05 ± 0.16	61.91 ± 0.16
DC + IMP (ours)	-	**57.63 ± 0.23**	**62.91 ± 0.22**
PN [OLR18]	37.80 ± 0.40	53.30 ± 0.50	58.70 ± 0.40
TADAM [OLR18]	40.10 ± 0.40	56.10 ± 0.40	61.60 ± 0.50

Table 4.3 – *Average 5-way accuracy on novel classes of FC100 with ResNet-12.* The top part is our solutions and baselines. GAP: global average pooling (stage 1); DC: dense classification (stage 1); IMP: implanting (stage 2). In stage 2, we use GAP on both support and queries. At testing, we use GAP on support examples and GAP or DC on queries, depending on the choice of stage 1. The bottom part results are as reported in the literature. All experiments use the same ResNet-12.

to the entire base class training set for adaptation, which could be problematic in terms of storage and or computing cost, so it is not the standard few-shot learning setting. Additionally, this method is designed for classification into both base and novel classes together which is a more challenging task.

Finn et al. [FAL17] also propose to fine-tune the whole network using the few-shot data. Contrary to [QBL18], they use only the few-shot data for fine-tuning. This adaptation stage is replicated during the first training stage on the base dataset as they adopt a meta-learning method to train the embedding network. Adaptation is limited to a few gradient updates. Typically, they report few-shot classification results with three to ten gradient steps at test time. This small number of steps has two advantages. First, it allows them to integrate the adaptation process in their meta-learning framework. Second, they stop the adaptation before they can observe overfitting on the novel data. The difference is that in our case, we do not use meta learning to include the adaptation into the representation learning stage.

4.3.2 Method

In this section, using the validation set, we observe the effect of fine-tuning the last few layers of the embedding network on few-shot data. Contrary to the method introduced before, we do not try to control the overfitting in any way other than setting the number of steps.

Concretely, we use a prototype classifier with class prototypes $P := (\mathbf{p}_j)_{j=1}^{c'}$ which are obtained per class by averaging embeddings of support examples $\phi_\theta(\mathbf{x}')$ as defined by (2.12). The prototypes are updated each time θ is updated. The classifier $f_{\theta,P} : \mathcal{X} \rightarrow$

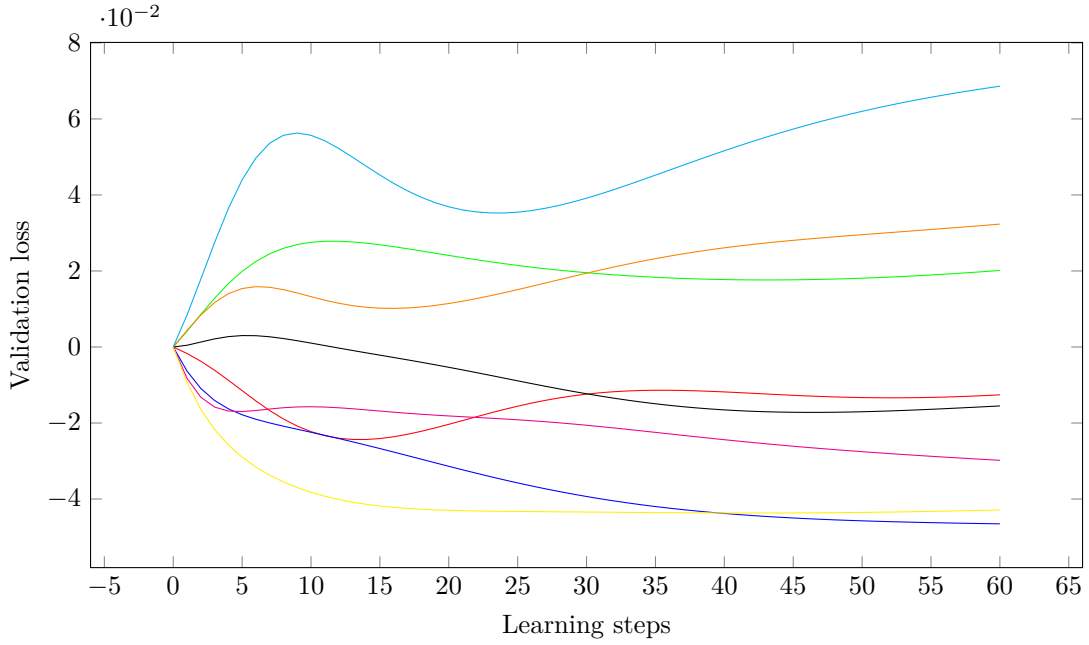


Figure 4.3 – 5-way 5-shot validation loss of *mini*Imagenet as a function of the number of learning steps at adaptation, relative to step 0 (no adaptation). Each curve corresponds to a different few-shot task build from the validation set.

$\mathbb{R}^{c'}$ is a standard cosine classifier (2.15) and the loss function is standard cross-entropy $J(X, \mathbf{y}; \theta, P)$ (2.17).

To test this method, we use the same embedding network as the one used for implanting, that is to say ResNet-12 trained with dense classification on the base dataset of *mini*Imagenet. For each task, we fine-tuned the embedding network for adaptation while keeping the first two residual blocks of the ResNet-12 fixed. We use Adam optimizer with learning rate $5e-5$ for 60 learning steps to monitor the evolution of accuracy and loss during adaptation.

In Figure 4.3 we show the evolution of the few-shot validation loss on eight 5-way 5-shot tasks built from the validation set of *mini*Imagenet. We observe that fine-tuning part of the embedding network is not a universal solution. For some tasks, the best validation loss is observed at iteration 0, that is to say without adaptation. For some other tasks, fine-tuning yields lower validation loss. In all cases, too many iterations leads to a drop in performance. In Figure 4.4 we show the evolution of the adaptation training loss on the same eight tasks. We observe that in all cases, the training loss is decreasing with more iterations. We conclude that the drop in validation is indeed due to overfitting.

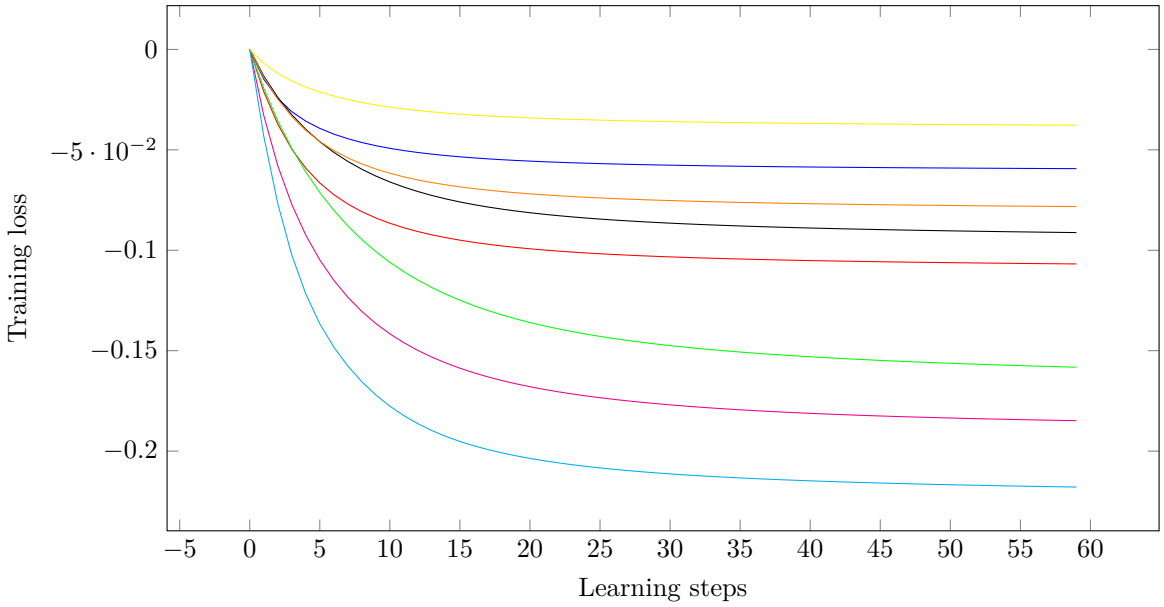


Figure 4.4 – 5-way 5-shot adaptation training loss of *mini*Imagenet as a function of the number of learning steps at adaptation, relative to step 0 (no adaptation). Each curve corresponds to a different few-shot task build from the validation set.

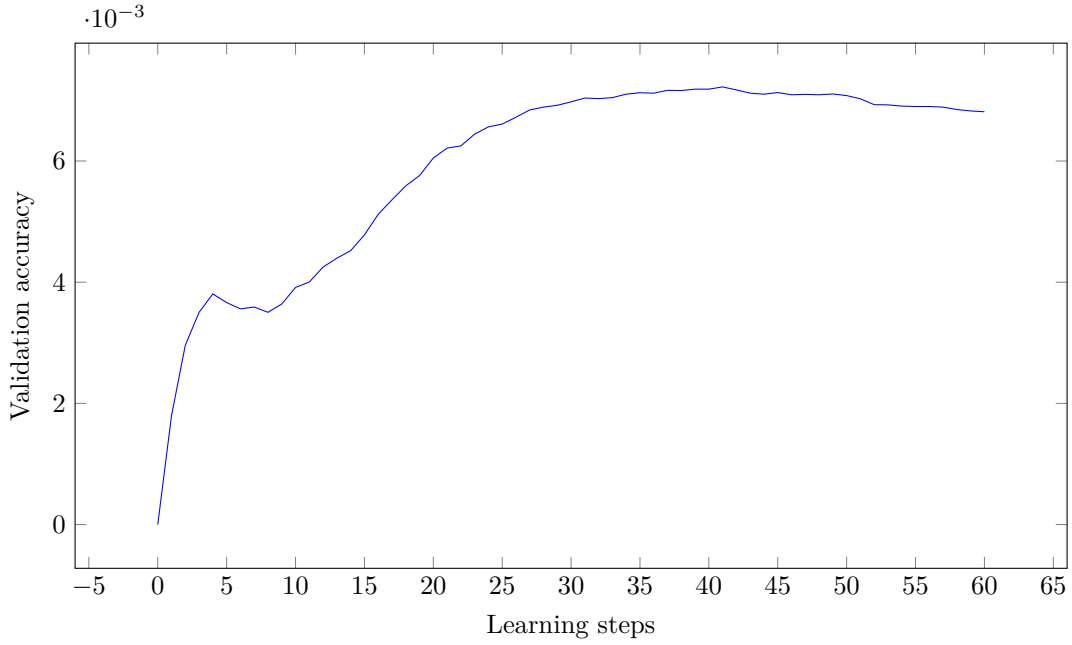


Figure 4.5 – 5-way 5-shot average validation accuracy of *mini*Imagenet as a function of the number of learning steps at adaptation, relative to step 0 (no adaptation).

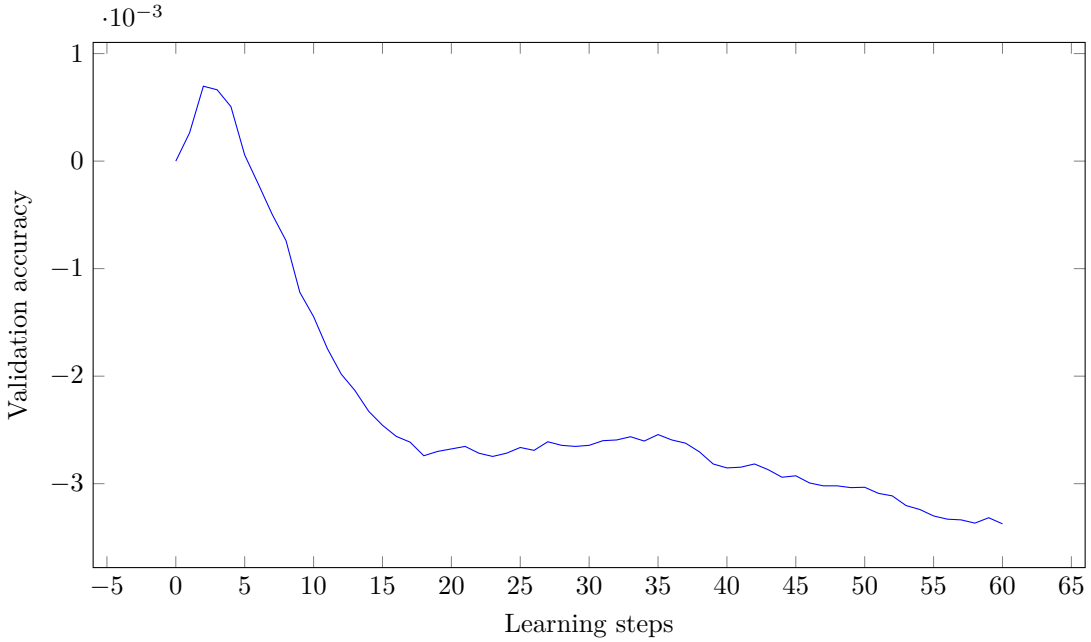


Figure 4.6 – 5-way 1-shot average validation accuracy of *mini*Imagenet as a function of the number of learning steps at adaptation, relative to step 0 (no adaptation).

We also display the behavior of the learning curves averaged for 2000 tested few-shot tasks sampled from the validation set of *mini*Imagenet. Figure 4.5 shows the average validation accuracy for 5-way 5-shot tasks. The peak of average accuracy is found at 41 steps of adaptation. Figure 4.6 shows the same average accuracy curve for 1-shot classification. Since the learning rate used is still $5e-5$, the peak is found earlier, at 2 steps of adaptation. Shortly after the peak, the accuracy decreases abruptly. This emphasizes the risk of overfitting learning with an extremely low amount of data. We performed other experiments with various numbers of residual blocks and learning rates which all resulted in lower accuracy or only marginally different to the one observed at peak on those curves.

When going to the test set, we cannot fairly obtain such curves. Indeed, finding the peak of the test accuracy and picking it would go against the core idea of having separate validation and test sets. Therefore, we use the optimal number of steps observed in the validation experiments as the number of learning steps to use at adaptation for the test tasks. This relies on the assumption that the tasks seen in the validation process are representative of the tasks built from the test set.

Method	1-shot	5-shot	10-shot
DC	62.25 ± 0.26	78.63 ± 0.19	82.78 ± 0.16
DC + IMP	-	79.77 ± 0.19	83.83 ± 0.16
DC + few-steps	**62.33 ± 0.26**	**79.87 ± 0.18**	**84.54 ± 0.15**

Table 4.4 – *Average 5-way accuracy on novel classes of* mini*ImageNet*, all on ResNet-12. DC: dense classification (representation learning stage); IMP: implanting (adaptation stage); few-steps: few-steps adaptation.

4.3.3 Results

Experimental setup Since we want to compare the few-steps adaptation solution to the implanting solution, we choose to use the same experimental setup. For adaptation, we train for 2, 41 and 55 steps respectively for 1-shot, 5-shot, and 10-shot. We show here the comparison on *mini*ImageNet. In chapter 6 we use few steps adaptation on an other network architecture and other datasets (specifically an upscaled version of *mini*Imagenet and CUB).

few-steps adaptation In Table 4.4 we show the test results of applying few-steps adaptation on *mini*ImageNet. In all cases, this method improves on the model without adaptation. In the case of 1-shot the improvement is marginal. However, the more support examples are available, the greater the gain from adaptation is, up to 1.7% increase of accuracy in the 10-shot case. This is consistent with the aforementioned relationship between training data and quality of learned model. In chapter 6 we oberve similar results on the other few-shot learning setting we introduce in this chapter.

Comparison with Implanting In Table 4.4 we show the test results from our model without any adaptation, with implanting adaptation and with few steps adaptation. Few steps adaptation results in a higher accuracy and can be used for one-shot as well as few-shot. As it is simplier and more versatile, we choose it as the preferred adaptation method in the following chapters. Implanting can still be useful in cases where it is impossible to determine an optimal number of learning steps to perform at adaptation (lacking a validation set or semantic gap between the validation set and the test set).

4.4 Using base classes to augment the support set

Overfitting during adaptation could also be limited if we could augment the support set. Augmentation is a common preprocessing stage when dealing with a computer vision task based on machine learning. It usually relies on simple transformations such as image deformations, or color jitter. With such strategies, the augmented examples are very similar to the original ones, no real new information about the class is added. Therefore, we cannot expect it to compensate for the lack of support examples in a few-shot learning task. A useful augmentation strategy for few-shot learning would generate new examples that are not trivial, adding knowledge about the intra-class variability of novel classes. Hallucination strategies have been proposed for this purpose [MMV00; HG17; Sch+18; Liu+19a]. They rely on learning a model, the hallucinator, which maps support examples to augmented versions. The hallucinator captures knowledge about intra-class variability from the base dataset, then uses this knowledge to generate new examples for the novel classes assuming a similar intra-class structure. In this section, we explore solutions for artificially augmenting the size of the support set. Instead of using complex hallucination models, we propose to select some related examples from the base dataset.

4.4.1 Method

The base dataset is usually discarded after the representation learning stage. Indeed, its only function is to train a task-specific model. Any adaptation method focuses on the novel classes where base classes are irrelevant. We argue that in many cases, base classes and novel classes are closely related. For instance, in the CUB dataset, classes representing subspecies of warblers are found in both base and novel datasets. While the subspecies are different, they share many common traits. We then propose a simple augmentation method that consists in picking related examples from the base dataset and using them as additional support examples.

More specifically, for each novel class i, we first compute its prototype P_i according to (2.12). Then, for each base class example x, we compute the similarity between its representation (after applying spatial pooling) $\phi_\theta(x)$ with P_i: $\cos(\phi(x), P_i)$. The top k_{extra} examples based on similarities are considered as closely related to the novel class, so we add them to the support set with class label i. In Figure 4.7, we show examples of the images selected by this method. Then, adaptation is performed by few gradient updates as explained above.

A drawback of this method is that we need to be able to access the base dataset at all

Figure 4.7 – Examples of selection of related base examples for a 5-way 1-shot task of CUB. Each row depicts the support image for the novel class (left) and the corresponding closest three examples in the base dataset based on cosine similarity in the feature space (right).

Base	*mini*ImageNet	CUB
DC	63.13 ± 0.28	74.97 ± 0.29
DC + Few step	63.66 ± 0.27	75.07 ± 0.29
DC + Few step + Augmentation	**65.84 ± 0.28**	**76.78 ± 0.29**

Table 4.5 – *Average 5-way 1-shot accuracy on novel classes of CUB and* mini*ImageNet. Augmentation is performed by adding 100 extra examples from the base dataset per novel class. ResNet-12 is used as embedding network.*

times, which could cause storage problem. Additionally, the selection of examples and adaptation with the added examples makes solving a new few-shot task more expensive. We limit those drawbacks by precomputing intermediate representations of all images from the base dataset. In particular, global representations are stored to accelerate the selection process. Intermediate representations given by the first blocks of the embedding network are also stored as those layers get frozen during adaptation. In parallel to our work, Afrasiyabi *et al.* [ALG19] also proposed adaptation by combining support examples to related base examples. They select base classes that are similar to the novel ones and then encourage the representation of the examples from the related classes to be close to the one of the support examples during adaptation.

4.4.2 Results

We applied this simple method of augmentation of the support set on CUB and an upscaled version of *mini*ImageNet. In Table 4.5, we show the results for 1-shot classification. We observe that for one-shot learning, this simple method is very effective. The more examples we add to the support set, the better the final average accuracy is. We report results with 100 extra examples per class, which corresponds to a plateau in terms of accuracy. However, in the case of few-shot learning with multiple support examples, this method seems to fail. We do not report those as in all cases adapting with the mix of support and extra examples results in lower average accuracy than the model without adaptation. Note that the results for *mini*ImageNet without augmentation are not the same as the ones reported in Table 4.4 as we are using a version of *mini*ImageNet with higher image resolution.

4.5 Conclusion

We have seen in this chapter that it is indeed possible to adapt the embedding network using only the few-shot data. We presented two solutions for such adaptation. First, *implanting*, which focuses on learning a few task-specific extra parameters, called implants, on the novel data. We introduce a way to train the implants involving building subtasks from the support examples. Another method introduced here is *few-steps adaptation*. It consists in simply fine-tuning part of the embedding network for a limited number of training steps. This early stopping strategy is based on the observed validation accuracy peak computed with validation few-shot tasks. Although simple, this method is able to improve significantly the performance of our model on 5-way 5-shot tasks. On one-shot tasks, it is more difficult to stop before overfitting, resulting in only small improvements compared to no adaptation. A way to make few-steps adaptation more impactful in the one-shot case is to select relevant examples from the base classes by comparing their representations with the support examples. While simple, this augmentation strategy shows promising results.

Chapter 5

Local propagation for transductive few-shot learning

Contents

The core problem of few-shot learning is the lack of data coming from the novel classes. We have discussed in the previous chapters on how we can artificially augment the novel data in order to directly tackle the issue. Generation based methods generate

new novel class data by using the base dataset. This is not ideal since there might be a semantic gap between the base and novel classes. On top of that, base class data is already used for representation learning as seen in chapter 3, so using it again in for classification into the novel classes might be redundant.

One source of data that is not used in the usual few-shot learning formulation is the queries themselves. In the case where we are presented with multiple queries to classify, we can consider them as a set of unlabeled data from the novel classes. This different few-shot setup is called *transductive few-shot learning*. In this case, learning to classify among the novel classes becomes similar to the well studied *semi-supervised learning* task. In semi-supervised learning, the training set is only partially labeled. Some methods also aim at classifying the unlabeled images which is usually referred to as *transductive semi-supervised learning*. There are two main approaches to semi-supervised learning in the literature. On one hand, some methods guide the training process by adding an unsupervised loss to it [GB04; LA17; SJT16; TV17]. On the other hand, some methods assign pseudo labels to the unlabeled samples to include them in a supervised training process [Lee13; Shi+18; Isc+19a]. In the context of few-shot learning, supervised learning on the support examples is rarely attempted to avoid the risk of overfitting, therefore the first category of methods is not easily applicable. Inspired by the second category of methods, some few-shot learning works propose to build a graph with labelled and unlabeled examples as vertices, then propagate information from the labelled examples to the unlabeled ones. Liu *et al.* [Liu+19b] propagate label information, Rodriguez *et al.* [Rod+20] propagate features and Garcia *et al.* [GB18] learn the graph operation using a graph neural network.

Moreover, we argue that the global spatial pooling operation that is generally applied to the image embedding ignores the rich data that is hidden in each given example. Each image is inherently a collection of data, which has been exploited by *dense classification* at representation learning as explained in section 3.2 and *naïve Bayes nearest neighbor* (NBNN) [Li+19b] at inference.

In this chapter, we attempt to bridge these two ideas, *i.e.*, using a collection of query to classify as unlabeled data and using local representations of examples. We propose to learn a representation function using dense classification. Then for few-shot inference, we break down the convolutional activations of support and query images into pieces corresponding to different spatial positions, consider all these pieces as different examples, and then apply feature or label propagation [Rod+20] to these examples. Pieces originating in support examples inherit their labels as in dense classification and NBNN [Li+19b], while pieces originating in queries are unlabeled. Since there are a

number of unlabeled pieces per image, this gives rise to *transductive inference* even in the case of a single query image.

Propagation of information across local features of images could fail if we take into consideration irrelevant part of the image. For instance, the local features of the backgrounds of two different objects could be close to each other in the feature space, by having them both in the graph, we might incorrectly propagate information through the corresponding edge. Therefore, it is important to add to our method an attention mechanism, allowing to filter out some part of the image. Attention mechanisms in the feature space [Vin+16; Mis+18; Ren+18a; GK18; OLR18; Li+19a] have commonly been used in few-shot learning for a few years. However, studies on local information in images have only appeared more recently, following our work on dense classification and naïve Bayes nearest neighbor. Some works study spatial attention mechanisms [WH19; ZZK19; Xv+19], other focus on finding alignment between local features from the support and query set [Hou+19a; Hao+19; Wu+19; Zha+20]. In this work, we propose our own spatial attention mechanism. It is extremely simple and can be used at inference without any cost as it only uses our embedding network trained with dense classification in the representation learning stage. While being essential to our method, it also brings significant gains in all baselines were we applied it.

Local label propagation and the spatial attention mechanism introduced in this chapter have been published in [LAP20b].

5.1 Background

5.1.1 Transductive few-shot learning formulation

Transductive few-shot learning differs from the general few-shot learning definition as it considers multiple queries jointly and exploits their distribution, even though they are unlabeled. As introduced in section 2.2, we aim to classify query examples from $\mathcal{X}$ into the novel class set C'. Here we are given a set $Q := \{\mathbf{q}_i\}_{i=1}^{q}$ of query examples and a prediction is required for all queries in Q. In this case, although queries are unlabeled, we can take advantage of this additional data and learn a classifier f that is a function of both the labeled support data X' and the unlabeled queries Q. This transductive setting implies *semi-supervised learning*.

5.1.2 Label propagation

Many computer vision tasks can be addressed by forming a graph and propagating information on it. Usually graph vertices are distinct examples and edge values are similarities between their representations. Label propagation has been extensively researched in transductive inference. An overview of tranductive inference methods can be found in chapter 11 of [CSZ10]. In this case, vertices correspond to full images and label information is propagated from the labeled examples to the unlabeled ones. Another use is to segment semantically an image by building a graph where vertices are pixels.

Zhu and Ghahramani [ZG02] are the first to introduce an algorithm based on building a graph using both labeled and unlabeled examples to propagate label information. Since other similar algorithms have appeared with the same idea. [Zho+03a] is similar and is used in a few-shot learning context in [Liu+19b]. We choose to use this algorithm in our method.

Nodes of the graph are representations of the examples while edges are similarities between them. Graph are generally represented as a squared matrix W with one column (and row) for each node. Weight w_{ij} is the value given to the edge between vertices i and j. Edge values differ with the method used. A popular choice in this case is to use a Gaussian kernel [ZG02]:

$$W_{ij} = e^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}, \tag{5.1}$$

with x_i et x_j respectively the representations used for examples i and j and σ is the standard deviation of the kernel.

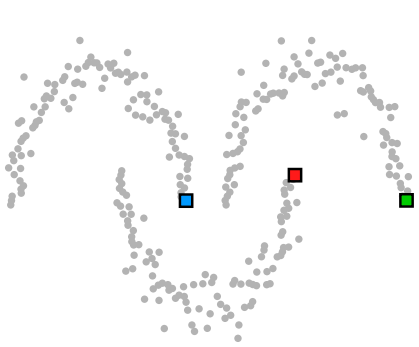
W is then symmetrically normalized:

$$\mathcal{W} := D^{-1/2} W D^{-1/2}, \tag{5.2}$$

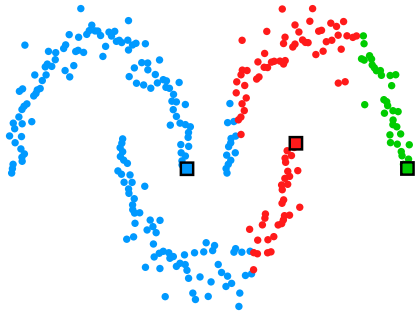
where $D := W\mathbf{1}_t$ is the *degree matrix* of the graph and $\mathbf{1}_t$ is the $t \times 1$ all-ones vector. Label information is encoded in matrix Y with one column per example and one line per class. For unlabeled examples, the corresponding column is all zeros, for labelled ones, the column is the one-hot encoded label vector. Labels are then propagated in an iterative fashion:

$$F(t+1) := \alpha F(t)\mathcal{W} + (1-\alpha)Y, \tag{5.3}$$

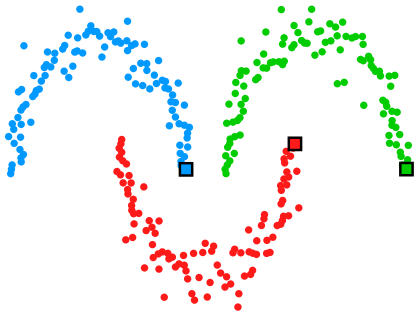
with $\alpha \in [0, 1)$ a parameter of the method. The first term of the sum corresponds to



(a) Partially labelled samples.



(b) Unlabeled samples take the label of the closest labelled sample.



(c) Unlabeled samples are labelled using label propagation.

Figure 5.1 – Illustration of labeling strategies of the unlabeled samples on a toy example. unlabeled samples are in grey. Labelled samples are colored squares, each color corresponding to a different class.

propagation of the label to its neighbors in the graph, as the updated class scores for a class will be a weighted average of the scores of its neighbors. The second term ensures that the scores remain close to the initial values given by Y. After convergence, examples can be labelled with the class whose score is the highest. Interestingly, they show that their iterative algorithm converges to a closed form solution:

$$\lim_{t\to\infty} F(t) = Y(1-\alpha)(I-\alpha\mathcal{W})^{-1}. \tag{5.4}$$

In Figure 5.1, we demonstrate on a toy example how label propagation can provide a good labeling of unlabeled samples. In this case there is only one labelled example per class but we can easily expand the behavior to multiple labelled examples per class.

In Figure 5.1b, we apply a simple labelling strategy where unlabeled samples take the label from the closest labeled samples. We observe that the resulting labelling is not satisfactory. Samples coming from a given class form a manifold, which is not captured by this labelling method. Such methods only relying on the similarity with labelled samples would only work if the representation of examples coming from the same class were all clustered closely to each other and separable from the other samples. Typically, that is what representation learning aims to do. However we cannot make those assumptions on the representation when the feature space has not yet been optimized to separate the classes or for transductive few-shot learning, where the feature space correctly is optimized to separate the base classes (which are labelled) but we have no guarantee on the novel classes. In contrast, label propagation captures the shape of the manifold of a class, resulting in a visually satisfying labelling, as in Figure 5.1c

5.2 Local features

In section 3.2 we discussed how we can consider spatial location of feature maps as local features of the image and how we can use those local features during the representation learning stage. Here we plan on using local features on the support examples X' of the novel classes. Because novel class data is lacking, it important in this stage to focus on only the relevant parts of the support examples. That is what lead us to experiment with spatial attention methods.

It is common in few-shot learning to use attention and adaptation mechanisms in the feature space [Vin+16; Mis+18; Ren+18a; GK18; OLR18; Li+19a]. However, despite being the subject of a pioneering work in 2005 [BU05], looking at local information in images has not been studied more recently in few-shot learning, until dense classification and naïve Bayes nearest neighbor [Li+19b]. We use the former for representation learning. The latter is similar to our work in using local representations at inference, the difference being that we apply propagation. These works have been followed by studies on *spatial attention* [WH19; ZZK19; Xv+19] and *alignment* [Hou+19a; Hao+19; Wu+19; Zha+20]. We experiment with an extremely simple spatial attention mechanism in this work, which requires no learning and boosts significantly all baselines.

5.2.1 Spatial attention

Before we can use the features of all spatial positions as data, it is important to suppress the background, which appears frequently across positions and images, without being

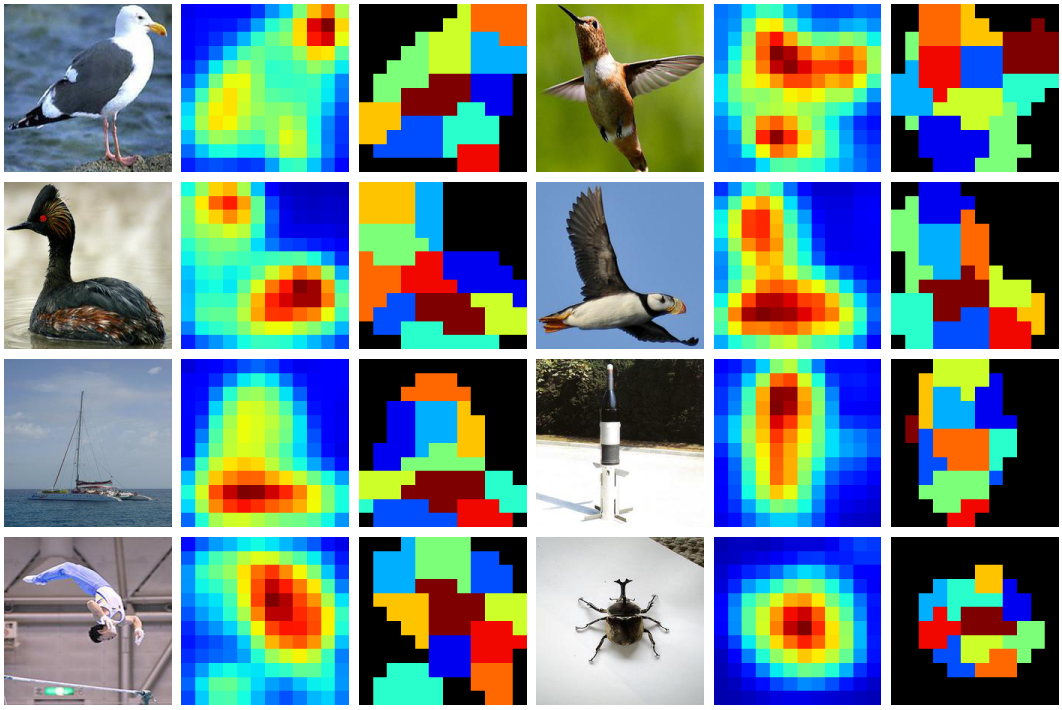


Figure 5.2 – Examples of images, each with the corresponding spatial attention heatmap and clusters used in feature pooling (black indicates regions below threshold in the heatmap). The first two lines correspond to CUB, the last two to *mini*ImageNet. We use $\tau = 0.3$ for spatial attention and $m = 10$ for feature pooling.

discriminative for the classification task. One approach to this problem is to learn a class-agnostic *spatial attention* mechanism [WH19; ZZK19]. We choose not to do that as it requires additional supervision. Indeed, [WH19] uses extra bounding box annotations to learn a detector and [ZZK19] use a weakly labeled image detector on data outside of the few-shot learning. Another approach is to study spatially the output feature channels. [KMO16] proposes a weighting mechanism to create powerful image representations for image search. One step of their method is to compute a spatial-wise weighting $S \in \mathbb{R}^{w \times h}$ for their feature maps:

$$S_{ij} = \left(\frac{S'_{ij}}{\left(\sum_{m,n} S'^{a}_{mn} \right)^{\frac{1}{a}}} \right)^{\frac{1}{b}} \tag{5.5}$$

where S'_{ij} is the l_1 norm of $\phi_\theta(x)$ at spatial location i, j, and a and b parameters of their methods. They show that this spatial weighting correctly focuses on the most relevant

parts of the image for image search. It can be explained intuitively as the network has learned to have a high output when coming across relevant information in the image for the task at hand. Similarly, Zhou *et al.* [Zho+16] also showed that high response in a particular spatial location indicates relevancy to the task. They propose to localize the relevant regions of the image by computing a *class activation map* as a weighted sum of the feature maps channels. In particular, their classifier is a fully-connected layer with no bias, and the weights for the feature maps are the values of the vector of this layer corresponding to the chosen class. This weighting is not applicable in our case as we aim to produce spatial attention maps that are class agnostic. Instead we will simply use the norm of the feature tensor at a particular position as evidence on the relevancy of the corresponding image region.

In particular, given an example $\mathbf{x} \in \mathcal{X}$ with feature tensor $F := \phi_{\theta'}(\mathbf{x})$, we select a subset of feature vectors $a(F) \subset \mathbb{R}^d$ at spatial positions $\mathbf{r} \in \Omega$ where the ℓ_2-norm is at least $\tau > 0$ relative to the maximum over the domain:

$$a(F) := \{F(\mathbf{r}) : \|F(\mathbf{r})\| \geq \tau \max_{\mathbf{t} \in \Omega} \|F(\mathbf{t})\|, \mathbf{r} \in \Omega\}. \tag{5.6}$$

Examples are shown in Figure 5.2. We find this mechanism particularly effective for its simplicity, not only for our method, but also for all baselines. No spatial attention is a special case where $\tau = 0$.

5.2.2 Feature pooling

Propagation tends to amplify elements that appear frequently in a dataset. Local propagation does the same for elements originating from different spatial positions, which in turn depends on the scale of objects relative to the spatial resolution. This can be particularly harmful with elements originating from background clutter and bypass condition (5.6), exactly because they appear frequently.

To obtain a fixed-size representation that only depends on the content, we perform pooling in the feature space into a fixed number of vectors per example. We do so by *clustering*: given an example $\mathbf{x} \in \mathcal{X}$ with selected feature vectors $a(\phi_\theta(\mathbf{x}))$ (5.6), we apply k-means on these vectors to cluster them into m clusters (notation k being used for k-shot classification). We obtain a set of m *feature centroids*, which we represent as columns in the $d \times m$ matrix $g_{\theta'}(\mathbf{x})$. Examples are shown in Figure 5.2. We use this representation only for local propagation. Global propagation and no feature pooling are special cases where $m = 1$ and $m = w \times h$ respectively.

5.3 Local propagation

Whatever is propagated (similarities, features, or labels), there are two extremes in graph-based propagation. At one extreme, vertices are *global* representations of images, and the graph represents a dataset. This can be used *e.g.* for similarity search [Zho+03b] or semi-supervised classification [ZG02; Zho+03a]. At the other extreme, vertices are *local* representations of pixels in an image, which can be used *e.g.* for interactive [Gra06; KLL08] or semantic [Ber+17] segmentation, or both [VC17]. *Regional representations* across images have been used for similarity search [Isc+17], but we believe we are the first to use *local* (pixel) or *semi-local* (clusters) representations across images for feature or label propagation.

5.3.1 General method

We develop this idea under the transductive setting because it is more general: The non-transductive is the special case where $q = 1$, the set of queries $Q = \{\mathbf{q}_1\}$ is singleton and we are making a prediction for $\mathbf{q}_1$. Given the support examples in X' and queries Q, we represent the feature centroids of both as columns in the $d \times t$ matrix

$$V := \begin{pmatrix} g_\theta(\mathbf{x}'_1) & \dots & g_\theta(\mathbf{x}'_n) & g_\theta(\mathbf{q}_1) & \dots & g_\theta(\mathbf{q}_q) \end{pmatrix} \tag{5.7}$$

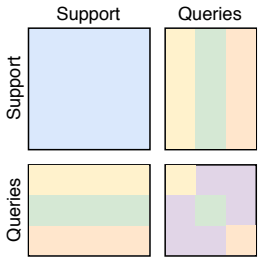
where $t := (n' + q)m$.

We use the pairwise similarity function to construct the nearest neighbor graph of the columns of V represented by the $t \times t$ symmetric non-negative *adjacency matrix* W_V with zero diagonal:

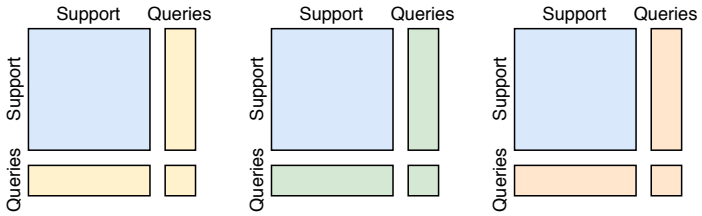
$$W_{ij} := \begin{cases} [\cos(\mathbf{v}_1, \mathbf{v}_2)]_+^\gamma, & \text{if } v_i \in \mathrm{NN}(v_j) \text{ or } v_j \in \mathrm{NN}(v_i) \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

where $\gamma > 1$ as in [Isc+17] and $\mathrm{NN}(v)$ refers to set of the nearest neighbors of v. We therefore consider all edges between two examples if one of them belongs to the nearest neighbors of the other. Another possibility that also gives a symmetric value is to only consider the mutual nearest neighbors for edges as it is done in [Isc+17]. In the context of our method, we experimented with the two options. We found similar results on our tasks, assuming the number of neighbors that we consider is well adapted, our method requiring a lower number of neighbors.

In terms of implementation, adding new unlabeled examples only requires computing similarities between the unlabeled examples and the previous examples. When evaluat-



(a) Nearest neighbor graph for transductive few-shot learning



(b) Nearest neighbors graphs for non-transductive few-shot learning. The blue part corresponding to similarities inside the support examples set is fixed and can be reused for new queries.

Figure 5.3 – Construction of the nearest neighbor adjacency matrix in the transductive (a) and non transductive (b) few-shot setting. For queries each color corresponds to local features coming from a common image.

ing few-shot performance with different sets of queries, used as a set in a transductive setting or as independent queries in a non-transductive setting, the part of the matrix W dedicated to support/support similarity can be computed only once as show in Figure 5.3. Additionally, the adjacency matrix for the non-transductive case can be deduced entirely from the transductive one with the same examples.

Following [Zho+03a], this matrix is symmetrically normalized as $\mathcal{W} := D^{-1/2} W D^{-1/2}$, where $D := W\mathbf{1}_t$ is the *degree matrix* of the graph and $\mathbf{1}_t$ is the $t \times 1$ all-ones vector.

Extending [Zho+03a], given any matrix $A \in \mathbb{R}^{u \times t}$ (or row vector for $u = 1$), its *propagation* on V is defined as

$$p_V(A) := A(1-\alpha)(I - \alpha\mathcal{W})^{-1}. \tag{5.9}$$

This is a smoothing operation on the graph of V, where parameter $\alpha \in [0, 1)$ controls the amount of smoothing: Columns of A corresponding to similar columns of V are averaged together. It is infinitely-recursive, as revealed by the series expansion of the matrix inverse [Zho+03a].

The operation (5.9) is called *local propagation* because the graph is defined on local representations originating from different spatial positions of the given images. *Global propagation* is the special case of having $m = 1$ cluster per image. This is the same as *global average pooling* (GAP), with or without spatial attention.

5.3.2 Local feature propagation

Using $A = V$, $u = d$ in (5.9), *local feature propagation* amounts to propagating V on itself:

$$\widetilde{V} := p_V(V), \tag{5.10}$$

in the sense that similar feature vectors in columns of V are averaged together, becoming even more similar. No feature propagation is a special case where $\alpha = 0$, $\widetilde{V} = V$.

5.3.3 Local label propagation

Given the propagated features $\widetilde{V}$ (5.10), we form a new graph with normalized adjacency matrix $\mathcal{W}_{\widetilde{V}}$. Extending [Zho+03a], we form the $c' \times t$ zero-one *label matrix* Y with one row per class and one column per spatial position over support examples and queries. A column corresponding to a spatial position of a support example x'_i is defined as the one-hot label vector $\mathbf{y}_i$; a column corresponding to a position of a query $\mathbf{q}_i$ is zero:

$$Y := \begin{pmatrix} \mathbf{y}'_1\mathbf{1}_m^\top & \dots & \mathbf{y}'_n\mathbf{1}_m^\top & \mathbf{0}_{c'\times m} & \dots & \mathbf{0}_{c'\times m} \end{pmatrix} \tag{5.11}$$

where $\mathbf{0}_{c'\times m}$ is the $c' \times m$ zero matrix and there are q such matrices. Using $A = Y$, $u = c$ in (5.9), *local label propagation* then amounts to propagating Y on $\widetilde{V}$:

$$\widetilde{Y} := p_{\widetilde{V}}(Y), \tag{5.12}$$

such that spatial positions with similar feature vectors obtain similar class scores. This may make little difference on labeled (support) examples, but is a mechanism for spatial positions of *unlabeled* (query) examples to obtain label information as propagated from spatial positions of labeled examples with similar features.

5.3.4 Inference

In $c' \times t$ matrix $\widetilde{Y}$ (5.12), there is one row per class and one column per spatial position over support examples and queries. $\widetilde{Y}$ is nonnegative; by column-wise ℓ_1-normalizing it into $c' \times t$ matrix $\hat{Y}$, we can interpret columns as probability distributions over classes per position. For each query example $\mathbf{q}_i$, if $\hat{Y}_i$ is the corresponding $c \times m$ submatrix of $\hat{Y}$, we average these distributions over positions, obtaining a distribution $\mathbf{p}_i := \hat{Y}_i\mathbf{1}_m/m$. Finally, as in (4.3), we make a discrete prediction $\pi(\mathbf{p}_i) = \arg\max_{j\in[c']} p_{ij}$ as the class of maximum probability. This operation is similar to NBNN (2.11), but the quantities being

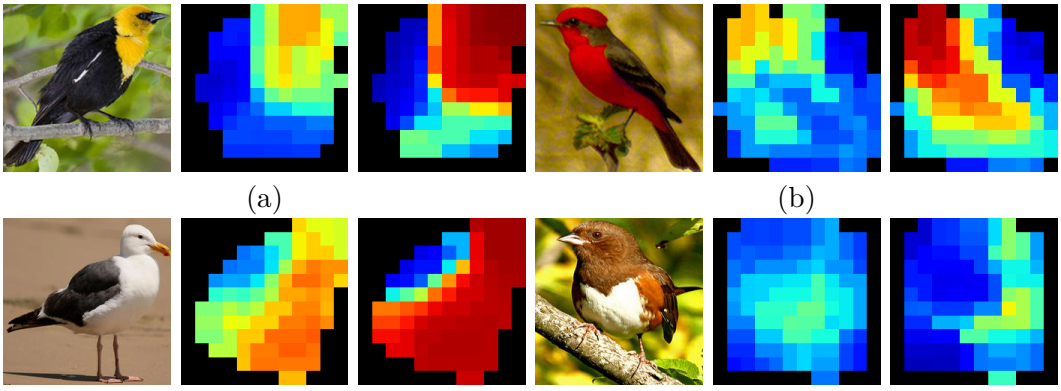


Figure 5.4 – Examples of CUB query images in 5-way 5-shot non-transductive tasks, each followed by the heatmap of predicted probability for the correct class using a prototype classifier, then using local label propagation. (a), (b) Local label propagation helps classifying to the correct class. (c) Both give a correct prediction. (d) Local label propagation fails.

averaged have undergone propagation rather than being direct similarities. Figure 5.4 shows examples of predicted probability for the correct class per spatial location. Local label propagation results in spatially smooth predictions that covers a large portion of the object.

5.4 Experiments

5.4.1 Experimental setup

Datasets We evaluate our method on CUB with images of resolution 224×224. We also experiment with *mini*ImageNet. Similarily to [Che+19; DSM19], we build a version of *mini*ImageNet with 224×224 images.

Network We test our method on a ResNet-12 embedding network. For input images of size 224×224, the embedding features are tensors of resolution 14×14. To adapt the the larger images before applying a dense classifier, we apply average pooling on these feature tensors, with kernel size 3×3 and stride 1 without padding. The resulting tensors are of resolution 12×12.

Base Training We train the network from scratch using stochastic gradient descent with Nesterov momentum on mini-batches of size 32. The learning rate schedule is set

according to the 5-way 5-shot validation accuracy.

Evaluation protocol For each dataset, we obtain a unique embedding network resulting from base training. All methods are then applied to the same features. For all experiments, we sample 2000 5-way few-shot tasks from the test set, each with 15 queries per class. We report the average accuracy as well as 95% confidence intervals. We evaluate two different settings: In the non-transductive setting, queries are treated as 75 distinct sets Q with only one query each, whereas in the transductive setting, there is a single set Q with all 75 queries.

Baselines In the *non-transductive* setting, we compare our method with variants of four existing few-shot inference methods. The first, referred to as GAP+Proto, applies *global average pooling* (GAP) on feature tensors and then uses a prototype classifier [SSZ17] on the support set (2.12). The second is the inference mechanism of the matching network [Vin+16], while the third, referred to as Local Match, is a modified version as follows. For each support example $\mathbf{x}'$ with feature tensor $F := \phi_\theta(\mathbf{x}')$, we use local feature vectors $F(\mathbf{r})$ at all positions $\mathbf{r} \in \Omega$ as independent support examples, with the same label as $\mathbf{x}'$. We do the same on queries and average the class score vectors over positions. The fourth is the inference mechanism of NBNN [Li+19b] (2.11). For each method, we experiment with and without our spatial attention mechanism (5.6). For Local Match and NBNN, we select a subset of local features per image. For GAP+Proto and Matching Net, we apply GAP to the selected subset only.

In the transductive setting, we compare with the inference mechanism of global label propagation [Liu+19c; Rod+20], with and without global embedding propagation [Rod+20]. These baselines are again evaluated with and without spatial attention. We always include spatial attention in our local propagation, but we experiment with and without feature pooling, with and without feature propagation.

5.4.2 Ablation studies

Overall, our method has five parameters. Two refer to optional components related to local information: the threshold τ for spatial attention and the number of clusters m for feature pooling. The other three refer to propagation, like all related methods dating back to [Zho+03a]: the number of neighbors in the graph, the exponent γ in the feature similarity function and α, controlling the amount of smoothing. For all experiments, we perform a fairly exhaustive parameter search over a small set of possible values per parameter and we make choices according to validation accuracy. We present a summary

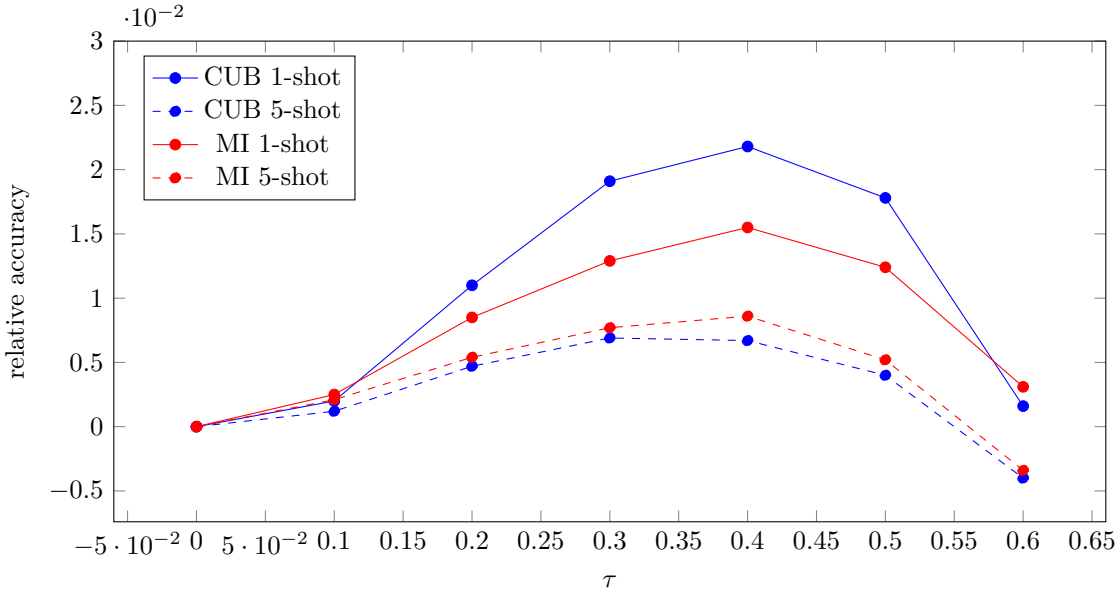


Figure 5.5 – *Spatial attention* on GAP+Proto [SSZ17]: 5-way few-shot classification accuracy *vs.* threshold τ, relative to $\tau = 0$ (no attention).

of parameter search independently for τ and m, keeping other parameters fixed to the optimal.

Spatial Attention As shown in Figure 5.5, referring to GAP+Proto baseline, there is an optimal range of τ in $[0.3, 0.5]$, such that we filter out the uniformative local feature without removing too much information. The same behavior appears in Figure 5.6, referring to our best method for each setting. For the remaining of the experiments, we fix τ to 0.3.

Feature pooling This is a compromise between global pooling and a full set of local features per image, which brings a consistent small improvement compared to both, while making local propagation more efficient by limiting the graph size. According to Figure 5.7, referring again to our best method for each setting, there is an optimal number m of clusters that depends on the dataset and setting (transductive or not, 1/5-shot). On CUB, we use $m = 40$ for 1-shot and $m = 60$ for 5-shot. On *mini*ImageNet, we use $m = 60$ in both cases.

Propagation parameters Propagation has been extensively researched in the past, so we do not report the study of its parameters. It is known for instance that α should

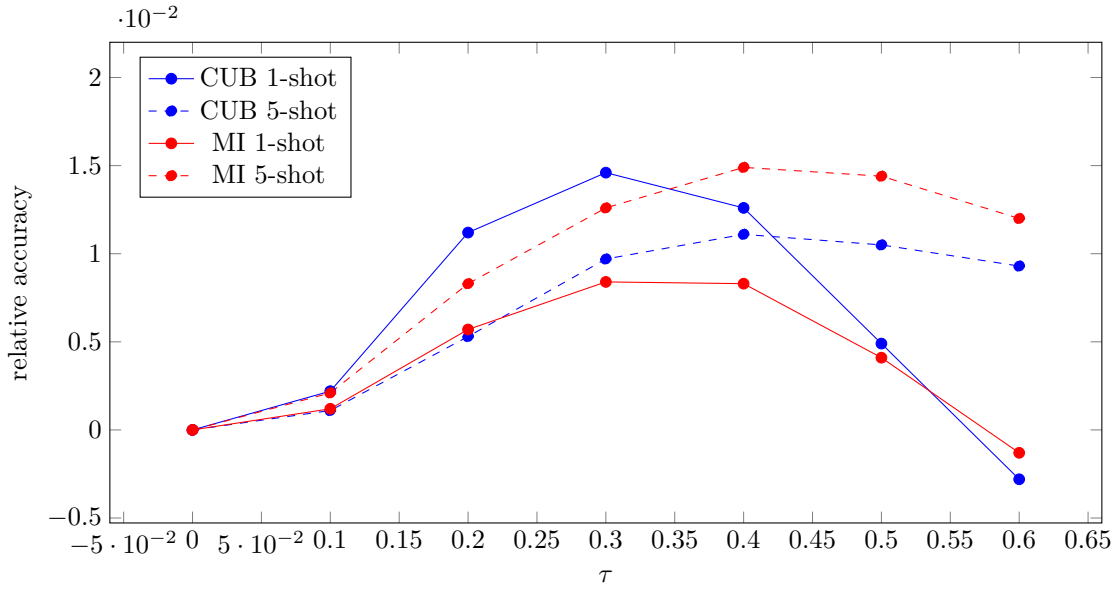


Figure 5.6 – *Spatial attention* on our local label propagation, including feature pooling and feature propagation: 5-way few-shot classification accuracy *vs.* threshold τ, relative to $\tau = 0$ (no attention). All other parameters fixed to optimal.

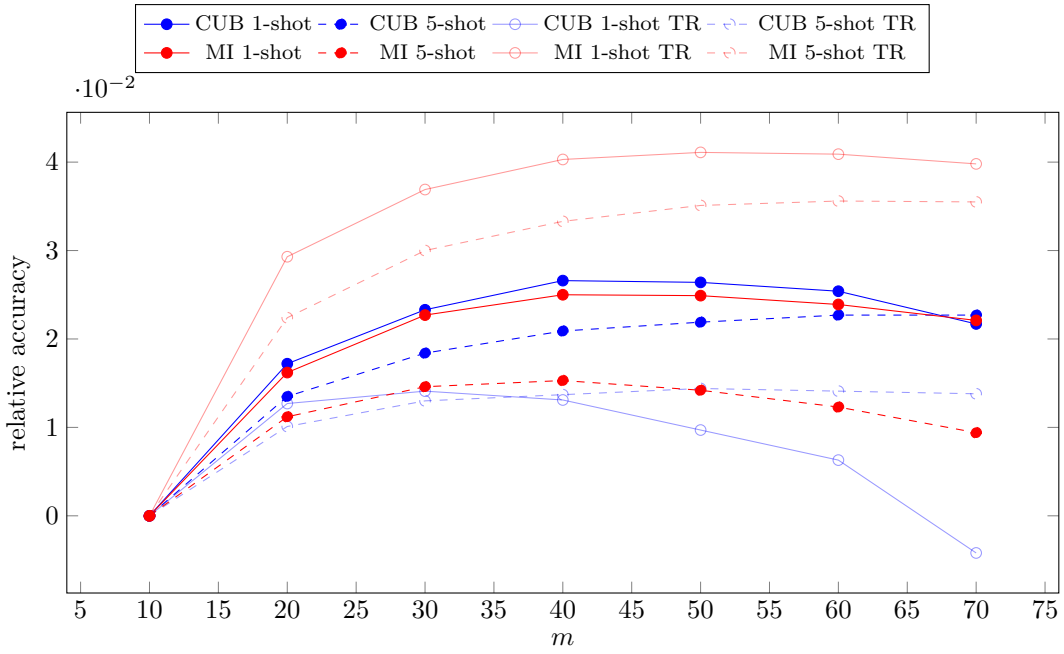


Figure 5.7 – *Feature pooling* on our local propagation: 5-way few-shot classification accuracy *vs.* number of clusters m, relative to $m = 10$ for better visualization. TR: transductive. We use spatial attention in all settings and feature propagation only in transductive. All other parameters fixed to optimal.

be close to 1 and there is a local maximum with respect to the number of neighbors to consider in the graph, which depends on the number of data [Isc+17]. After parameter search, for most experiments, we set $\alpha = 0.9$, $\gamma = 4$ and respectively 5 and 50 neighbors in the graph for global and local propagation

5.4.3 Results

Table 5.1 presents a complete set of results using our method and baselines in different settings, using different options. We discuss the effect of our contributions below.

Spatial attention We use spatial attention with our method but we also combine it with baselines for fair comparison. It is an extremely simple mechanism that consistently improves few-shot classification accuracy in most cases, including global or local, with propagation or not, transductive or not. The only exception is Local Match on *mini*ImageNet. The gain is more pronounced on 1-shot tasks, which is expected as information selection is more important when information is scarce. It reaches 3% for the baselines and 2% for propagation on CUB, as well as 1% on *mini*ImageNet.

Feature pooling Clustering the set of local features into a given number of clusters for each image is bringing small accuracy improvements when combined with propagation, local or global. In particular, spatial attention and feature pooling brings a 0.30% to 0.75% increase of accuracy compared to spatial attention alone on CUB (transductive and non-transductive). An exception is *mini*ImageNet non-transductive where feature pooling gives slightly worse accuracy by an insignificant margin.

Label propagation In the *non-transductive* setting, global label propagation fails. Its performance is similar or inferior to GAP+Protonet. This is to be expected, as this is method a transductive method, so it is not a natural choice given only one query. By contrast, our local label propagation succeeds even in this setting, with up to 2.7% improvement on CUB 5-way 1-shot compared to GAP+Proto. One exception is *mini*ImageNet 5-way 1-shot, where GAP+Proto is better by a small margin; in this case however, the other local baselines (Local Match and NBNN) are worse than both GAP+Proto and our local label propagation, by a larger margin.

In the transductive setting, label propagation, global or local, always helps by using unlabeled data. Our local label propagation with spatial attention and feature pooling improves 5-way 1-shot accuracy over the non-transductive setting by 6% and 5.5%, on CUB and *mini*ImageNet respectively. This improvement is lower for 5-shot tasks as

Method	A	P	F	CUB 1-shot	CUB 5-shot	*mini*ImageNet 1-shot	*mini*ImageNet 5-shot
GAP+Proto [SSZ17]				74.85±0.48	90.38±0.27	63.39±0.46	81.21±0.32
GAP+Proto [SSZ17]	✓			77.10±0.47	91.24±0.26	64.22±0.45	81.71±0.31
Matching Net [Vin+16]				74.85±0.48	89.23±0.29	63.39±0.46	78.14±0.33
Matching Net [Vin+16]	✓			77.10±0.47	89.95±0.28	64.22±0.45	78.70±0.33
Local Match [Vin+16]				75.92±0.46	89.16±0.28	64.05±0.46	78.45±0.34
Local Match [Vin+16]	✓			78.29±0.45	90.60±0.26	63.58±0.46	78.01±0.35
NBNN [Li+19b]				76.21±0.45	89.59±0.27	64.90±0.45	79.74±0.32
NBNN [Li+19b]	✓			79.14±0.44	91.40±0.25	65.18±0.45	80.00±0.31
Global Label Propagation, Non-Transductive							
Propagation				74.69±0.48	87.96±0.30	63.39±0.46	75.89±0.36
	✓			76.94±0.47	89.14±0.30	64.22±0.45	76.40±0.36
	✓		✓	77.23±0.46	88.78±0.31	63.41±0.45	77.04±0.37
Local Label Propagation (this work), Non-Transductive							
Propagation				78.24±0.44	91.07±0.26	65.52±0.45	80.49±0.31
	✓			79.02±0.44	91.81±0.25	**65.74±0.45**	**81.13±0.31**
	✓	✓		**79.77±0.44**	**92.07±0.25**	65.59±0.45	80.73±0.31
	✓	✓	✓	79.32±0.44	91.52±0.25	64.43±0.45	80.26±0.32
Global Label Propagation, Transductive							
Propagation				83.64±0.48	90.63±0.27	70.07±0.51	80.96±0.34
	✓			85.52±0.46	91.67±0.27	70.67±0.51	81.44±0.33
	✓		✓	87.18±0.46	91.88±0.27	72.54±0.54	81.38±0.35
Local Label Propagation (this work), Transductive							
Propagation				83.04±0.43	91.89±0.25	69.95±0.48	82.13±0.31
	✓			85.33±0.42	92.50±0.25	71.00±0.48	**82.87±0.30**
	✓	✓		85.80±0.41	92.92±0.24	71.12±0.48	82.83±0.31
	✓	✓	✓	**87.77±0.41**	**93.35±0.23**	**72.57±0.51**	82.76±0.33
Other models, Non-Transductive							
SNAIL [Mis+18]				-	-	55.71±0.99	68.88±0.92
TADAM [OLR18]				-	-	58.50±0.30	76.70±0.30
DC+IMP [Lif+19]				-	-	62.53±0.19	79.77±0.19
Neg-Cosine [Liu+20]				-	-	62.33±0.82	80.94±0.59
Other models, Transductive							
TPN [Liu+19b]				-	-	59.46	75.65
LR+ICI [Wan+20b]				88.06	92.53	66.80	79.26
EPNet [Rod+20]				82.85±0.81	91.32±0.41	66.50±0.89	81.06±0.60

Table 5.1 – 5-way few-shot classification accuracy, comparing our local (feature and label) propagation to baselines and existing work. A: spatial attention (our work, also applied to baselines). P: feature pooling (clustering) (our work). F: feature propagation [Rod+20].

more labeled data are used. Compared with global label propagation, it improves by up to 1.5% on 5-shot, CUB and *mini*ImageNet.

Feature propagation In the *non-transductive* setting, feature propagation is mostly harmful, especially when used with our local label propagation, which remains the best option, together with feature pooling. In the transductive setting however, it helps both global and local label propagation, the only exception being 5-shot, *mini*ImageNet. In the case of local label propagation with feature pooling, the gain is up to 2% and 1.5% on 1-shot, CUB and *mini*ImageNet respectively. Therefore this combination is the most effective, improving over our best non transductive result by 8% and 6.5% on 1-shot, CUB and *mini*ImageNet respectively.

Universality As shown in Figure 5.8, our local label propagation is a *universally safe choice* for few-shot inference under both transductive and non-transductive settings. This contrasts with existing methods such as global label propagation, where the user needs to make decisions depending on the amount of unlabeled data that is available.

Comparison to existing methods Table 5.1 also includes a number of recent few-shot learning methods. For fair comparison, all reported results are using the same ResNet12 as embedding network. We observe that our baseline GAP+Proto is better than these models on non-transductive 5-shot classification on *mini*ImageNet. Our method is then outperforming those models as well. In the transductive setting, global propagation is weaker than existing methods, but our best setting of local propagation (including spatial attention, feature pooling, feature propagation, and label propagation) is stronger in general. The only exception is 1-shot classification on CUB, where LR+ICI [Liu+19b] is stronger by a small margin.

In parallel with this work, two methods appeared very recently, which are stronger than our solution on *mini*ImageNet but weaker on CUB: (1) DGPN [Yan+20], which is yet another graph-based method and could be easily integrated with our local propagation. (2) DeepEMD [Zha+20], which is based on pairwise image alignment. This is more challenging to integrate, for instance, one would need to use alignment in the definition of the graph itself. This can be interesting future work.

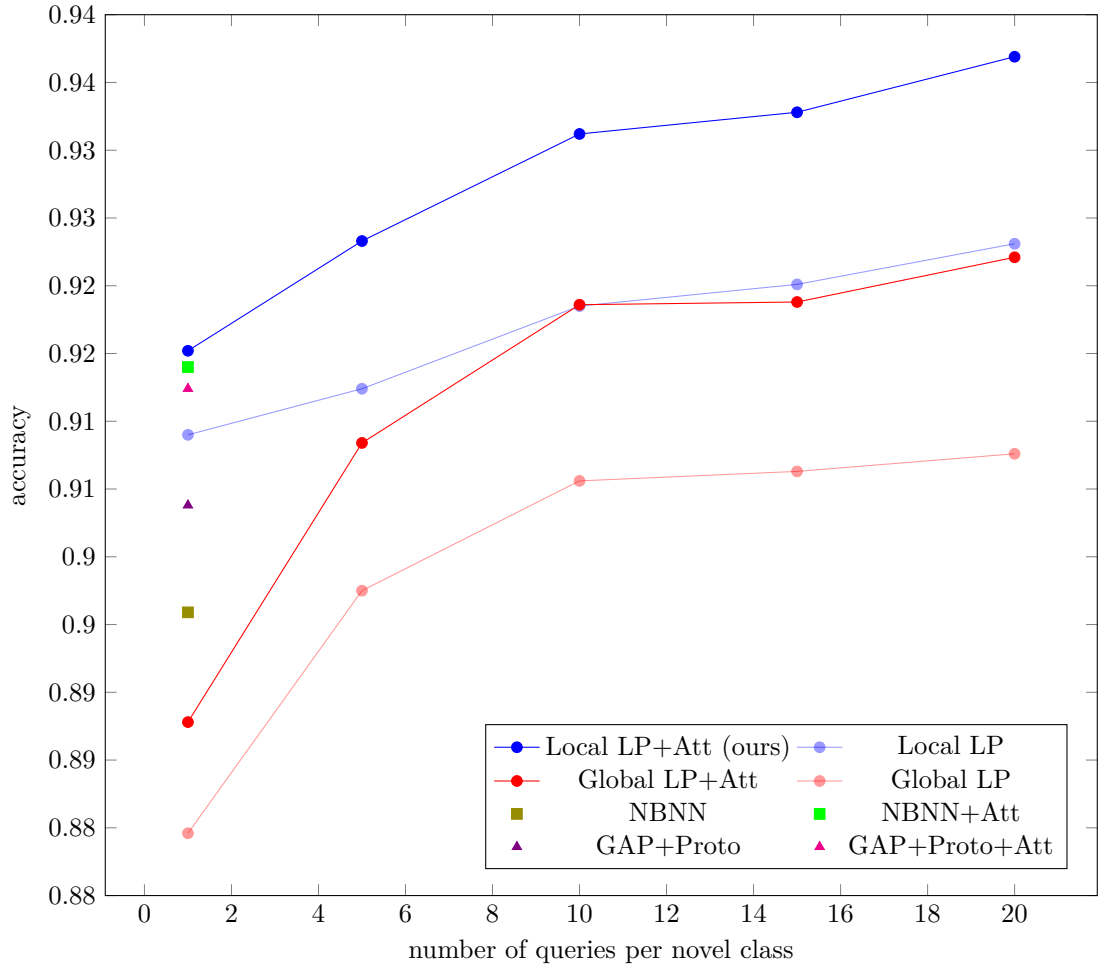


Figure 5.8 – CUB 5-way 5-shot classification accuracy *vs.* number of queries per novel class. Our local label propagation outperforms transductive and non-transductive baselines in all settings. By contrast, global label propagation only competes with non-trasductive methods when at least 10 unlabeled queries are available. We use spatial attention (also our contribution) and feature propagation [Rod+20] for all methods. We use feature pooling for local propagation

5.5 Conclusion

We have seen that when possible, considering queries as a set of unlabeled data to classify all at once can leave to improved classification performance. One way to take advantage of this is to build a graph with both labelled and unlabeled examples and propagate label information on it. Another way to artificially augment the size of the available data is to consider local features of the images.

Our *local propagation* framework takes the best of both worlds: more data from local representations and better use of this data from propagation. It provides a unified solution that works well given few labeled data and an arbitrary number of unlabeled data. As a result, it works better that solutions meant for the standard few-shot inference and at the same time better than solutions meant for transductive few-shot inference. Two secondary contributions are extremely simple and effective: (a) our *feature pooling* helps control the additional cost related to local features, while improving performance in most cases; (b) our *spatial attention* helps not only our method but all baselines too, by a significant margin on 1-shot classification. Our solution only affects inference, so it can easily be plugged into any alternative representation learning method. It is general enough to integrate other state-of-the-art solutions, like pairwise image alignment, other forms of propagation and propagation on several layers.

Chapter 6

Few-shot Few-shot learning

Contents

As introduced before, few-shot learning aims at replicating the ability of humans to learn to solve new tasks from a small amount of training examples. However, there is a considerable gap between this motivation and how the few-shot classification task is classically set up. On the one hand, for the sake of simplicity in experiments, the base class dataset where the representation is learned from scratch, contains a few dozen or hundred classes with a few hundred examples each. This is by no means comparable to

datasets available to date [Kuz+18; Mah+18], let alone the amount of prior knowledge that a human may have accumulated before learning a new task. On the other hand, for a given domain of novel classes, *e.g.* bird species [Wel+10], base class data of such size in the same domain may not exist.

In this chapter, we depart from the standard few-shot classification scenario in two directions. First, we allow the representation to be learned from a large-scale dataset in a domain different than the base and novel class domain. In particular, we model prior knowledge with a classifier that is pre-trained on such a dataset, having no access to its training process. We thus maintain the difficulty of domain gap and the simplicity of experiments (by not training from scratch), while allowing a powerful representation. Second, we assume only few or zero examples per base class. Hence, the role of base classes is to adapt the representation to the domain at hand rather than learn from scratch. This scenario can be seen as few-shot version of few-shot learning. Hence, we call it *few-shot few-shot learning*.

We treat this problem as a two-stage adaptation process, first on the few base class examples if available and second on the even fewer novel class examples. Because of the limited amount of data, it is not appropriate to apply *e.g.* transfer learning [Ben+11] or domain adaptation [GL14], in either of the two stages. Because the network is pre-trained, and we do not have access to its training process or data, meta-learning is not an option either. We thus resort to few steps of fine-tuning as in the meta-testing stage of Finn *et al*. [FAL17] and Ravi and Larochelle [RL17] and chapter 4.

Focusing on image classification, we then investigate the role of spatial attention in the new problem. With large base class datasets, the network can implicitly learn the relevant parts of the images to focus on. In our setup, base class data are few, so our motivation is that a spatial attention mechanism may help the classifier in focusing on objects, suppressing background clutter. We observe that although the prior classes of the pre-trained classifier may be irrelevant to a new task, uncertainty over a large number of such classes may express anything unknown like background. This is a class-agnostic property and can apply to new tasks.

In particular, given an input image, we measure the entropy-based *certainty* of the pre-trained classifier in its prediction on the prior classes at every spatial location and we use it to construct a spatial attention map. This map can be utilized in a variety of ways, for instance, weighted spatial pooling or weighted loss per location; and in different situations like the two adaptation stages or at inference. We show that this spatial attention mechanism is more suitable for few-shot few-shot learning than the mechanism introduced in chapter 5. By using it, we are able to easily adapt a pre-

trained network to novel classes, without meta-learning.

Few-shot few-shot learning and the spatial attention mechanism introduced in this chapter have been published in [LAP20a].

6.1 Few-shot Few-shot formulation

The problem we consider is a variant of few-shot learning that has not been studied before. It involves sequential adaptation of a given network in two stages, each comprising a limited amount of data. There are many ways of exploiting prior learning to reduce the required amount of data and supervision like *transfer learning* [Ben+11; Yos+14], *domain adaptation* [GL14; RBV17; MDL18], or *incremental learning* [LH18; RKL16; Yoo+18]. However, none applies to the few-shot domain where just a handful of examples are given. In this section, we formally introduce the problem and its notations.

6.1.1 Reminder

In the usual few-shot scenario we are given a set of *training examples* $X := \{\mathbf{x}_i\}_{i=1}^n \subset \mathcal{X}$, and corresponding labels $\mathbf{y} := (y_i)_{i=1}^n \subset C^n$ where $C := [c] := \{1, \dots, c\}$ is a set of *base classes*. The objective is to learn a representation on these data, a process that we call *base training*, such that we can solve new tasks. A new task comprises a set of *support examples* $X' := \{\mathbf{x}'_i\}_{i=1}^{n'} \subset \mathcal{X}$ and labels $\mathbf{y}' := (y'_i)_{i=1}^{n'} \subset (C')^{n'}$, where $n' \ll n$ and $C' := [c']$ is a set of *novel classes* disjoint from C. The most common setting is k examples per novel class, so that $n' = kc'$, referred to as *c'-way, k-shot* classification. The objective now is to learn a classifier on these support data, a process that we call *adaptation*. This classifier should map a new *query example* from $\mathcal{X}$ to a prediction in C'.

6.1.2 Few-shot few-shot classification

Few-shot classification assumes that there is more data in base than novel classes, and a domain shift between the two, in the sense of no class overlap. Here we consider a modified problem where n can be small or *zero*, but there is another set $C^\circ = [c^\circ]$ of *prior classes* with even more data X° and labels $\mathbf{y}^\circ$ with $n \ll n^\circ := |X^\circ|$ and a greater domain shift to C, C'. Again, the most common setting is k' examples per base class, so that $n = k'c$. We are using a classifier that is *pre-trained* on this data but we do not have direct access to either $X^\circ, \mathbf{y}^\circ$, or its learning process. The objective of base

training is now to adapt the representation to the domain of C, C' rather than learn it from scratch; but we still call it *base training*.

6.2 Spatial attention from pre-training

We have seen in chapter 5 that filtering out the background information of the images is essential to our local label propagation method as it prevents propagation of labels through similar irrelevant parts of the images. In the few-shot few-shot learning problem, such issue does not appear. However, with few or even no base class images to learn on, the embedding network cannot learn to focus on the parts of the images that are relevant to the classification of images in the base and novel set domain. Therefore, albeit for different motivation, spatial attention is useful for few-shot few-shot learning as well. In this section, we propose a novel spatial attention mechanism that takes advantage of the pre-trained classifier used in this setup. In order to differentiate between the two attention mechanisms, we refer to the spatial attention from the previous chapter as norm spatial attention (NSA) and the one introduced here as entropy spatial attention (ESA).

6.2.1 Generating the attention weights

We assume a pre-trained network with an embedding function $\phi_{\theta^\circ} : \mathcal{X} \to \mathbb{R}^{r \times d}$ followed by *global average pooling* (GAP) and a classifier that is a fully connected layer with weights $W^\circ := (\mathbf{w}_j^\circ)_{j=1}^{c^\circ} \in \mathbb{R}^{d \times c^\circ}$ and biases $\mathbf{b}^\circ \in \mathbb{R}^{c^\circ}$, denoted jointly by $U^\circ := (W^\circ, \mathbf{b}^\circ)$. Without re-training, we remove the last pooling layer and apply the classifier densely as in 1×1 convolution, followed by softmax with temperature T. Then, similarly to (3.1), the classifier $f_{\theta^\circ, U^\circ} : \mathcal{X} \to \mathbb{R}^{r \times c^\circ}$ maps an example to a vector of probabilities per location, where classifier parameters U° are shared over locations:

$$f_{\theta^\circ, U^\circ}(\mathbf{x}) := \left[\boldsymbol{\sigma} \left(\frac{1}{T} \left(W^{\circ\top} \phi_{\theta^\circ}^{(q)}(\mathbf{x}) + \mathbf{b}^\circ \right) \right) \right]_{q=1}^{r}. \tag{6.1}$$

We now want to apply this classifier to examples in the set X (resp. X') of base (resp. novel) classes C (resp. C') in order to provide a *spatial attention* mechanism to embeddings obtained by parameters θ (resp. θ'). We formulate the idea on X, C, θ in this section but it applies equally to X', C', θ'. In particular, given an example $\mathbf{x} \in X$, we use the vector of probabilities $\mathbf{p}^{(q)} := f_{\theta^\circ, U^\circ}^{(q)}(\mathbf{x})$ corresponding to spatial location $q \in [r]$ to compute a scalar weight $w^{(q)}(\mathbf{x})$, expressing the *discriminative power* of the

Figure 6.1 – Examples of images from CUB (top) and *mini*ImageNet (bottom) overlaid with entropy-based spatial attention maps obtained from (6.2) using only the predicted class probabilites from ResNet-18 pre-trained on Places. See section 6.4 for details on datasets and networks.

particular location q of example $\mathbf{x}$.

Since $\mathbf{x}$ belongs to a set of classes C different than C°, there is no ground truth to be applied to the output of the pre-trained classifier f_{θ°,U°. However, the distribution $\mathbf{p}^{(q)}$ can still be used to evaluate how discriminative the input is. We use the entropy function for this purpose, $H(\mathbf{p}) := -\sum_j p_j \log(p_j)$. We map the entropy to $[0, 1]$, measuring the *certainty* of the pre-trained classifier in its prediction on the prior classes C°:

$$w^{(q)}(\mathbf{x}) := 1 - \frac{H(f^{(q)}_{\theta^\circ,U^\circ}(\mathbf{x}))}{\log c^\circ} \tag{6.2}$$

for $q \in [r]$, where we ignore dependence on parameters θ°, U° to simplify the notation, since they remain fixed. We use this as a weight for location q assuming that uncertainty over a large number of prior classes expresses anything unknown like background, which can apply to a new set of classes. We then ℓ_1-normalize the weights $w(\mathbf{x}) := [w^{(q)}(\mathbf{x})]_{q=1}^r \in \mathbb{R}^r$ as $\hat{w}(\mathbf{x}) := w(\mathbf{x}) / \|w(\mathbf{x})\|_1$. We call $\hat{w}(\mathbf{x})$ the *spatial attention weights* of $\mathbf{x}$.

Figure 6.1 shows examples of images with spatial attention maps. Despite the fact that there has been no training involved for the estimation of attention on the particular datasets, the result can still be useful in suppressing background clutter.

6.2.2 Applying the attention weights

If the embedding $\phi_\theta(\mathbf{x})$ is normally a vector in $\mathbb{R}^d$ obtained by GAP on a feature tensor $\Phi_\theta(\mathbf{x}) \in \mathbb{R}^{r\times d}$ as $\frac{1}{r}\sum_{q\in[r]} \Phi_\theta^{(q)}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$, then GAP is replaced by global *weighted* average pooling (GwAP):

$$\phi_\theta(\mathbf{x}) := \sum_{q\in[r]} \hat{w}^{(q)}(\mathbf{x})\Phi_\theta^{(q)}(\mathbf{x}). \tag{6.3}$$

for $\mathbf{x} \in \mathcal{X}$. We recall that this applies equally to θ' in the case of novel classes.

It is also possible to combine ESA with NSA from chapter 5. In this case, NSA is used first, then the spatial attention weights are computed for the remaining spatial positions before pooling.

6.3 Few-shot few-shot classification model

In this section, we describe stage by stage our framework for few-shot few-shot learning, which include using the attention maps at inference on novel classes, as well as learning on novel classes. In the latter case, the weights are pre-computed for all training examples since the pre-trained network remains fixed in this process. In summary, we either replace GAP by GwAP (6.3) in all inputs to the embedding network, or use dense classification (3.1).

6.3.1 Base class training

Starting from a pre-trained embedding network ϕ_{θ°, we can either solve new tasks on novel classes C' directly, in which case $\theta = \theta^\circ$, or perform base class training, fine-tuning θ from θ°. Adaptation may involve for instance fine-tuning the last layers or the entire network, applying a spatial attention mechanism or not. Recalling that ϕ_{θ° is still needed for weight estimation (6.2), the most practical setting is to fine-tune the last layers, in which case ϕ_θ shares the same backbone network with ϕ_{θ°. Following MAML [FAL17], we perform a few gradient descent steps with low learning rate.

We use a dense classifier $f_{\theta,W} : \mathcal{X} \to \mathbb{R}^{r\times c}$ (3.1) with class weights W. Given the few base class examples X and labels $\mathbf{y}$, we learn W at the same time as fine-tuning θ by minimizing (3.2).

6.3.2 Novel class adaptation

Optionally, given the few novel class support examples X' and labels $\mathbf{y}'$, we can further adapt the embedding network, while applying our attention mechanism to the loss function. As in subsection 6.3.1, $\phi_{\theta'}$ shares the same backbone with ϕ_θ, being derived from it by fine-tuning the last layers. We perform even fewer gradient descent steps with lower learning rate

We use a prototype classifier where vector embeddings $\phi_{\theta'}(\mathbf{x}')$ of support examples $\mathbf{x}' \in X'$ are obtained by GwAP with $\phi_{\theta'}$ defined as in (6.3) and class prototypes $P := (\mathbf{p}_j)_{j=1}^{c'}$ are obtained per class by averaging embeddings of support examples as defined by (2.12) and updated whenever θ' is updated. The classifier $f_{\theta',P} : \mathcal{X} \to \mathbb{R}^{c'}$ is a standard cosine classifier (2.15) and the loss function is standard cross-entropy $J(X, \mathbf{y}; \theta, P)$ (2.17) with embedding $\phi_{\theta'}$ obtained by GwAP (6.3). Attention weights apply to embeddings of all inputs to the network, each time focusing on most discriminative parts. In case of no adaptation to the embedding network, we fix $\theta' = \theta$. Computing the prototypes P is then the only learning to be done and we can proceed to inference directly.

6.3.3 Novel class inference

At inference, as in subsection 6.3.2, we adopt a prototype classifier where vector embeddings $\phi_{\theta'}(\mathbf{x}')$ of support examples $\mathbf{x}' \in X'$ are obtained by GwAP with $\phi_{\theta'}$ defined as in (6.3) and class prototypes $P := (\mathbf{p}_j)_{j=1}^{c'}$ are obtained per class by averaging embeddings of support examples as defined by (2.12). Then, given a query $\mathbf{x} \in \mathcal{X}$, we similarly obtain a vector embedding $\phi_{\theta'}(\mathbf{x})$ by GwAP (6.3) and predict the class $\pi(f_{\theta',P}(\mathbf{x}))$ of the nearest prototype according to cosine similarity where π is given by (4.3) and $f_{\theta',P}$ by (2.15). We thus focus on the discriminative parts of both support and query examples, suppressing background clutter.

6.4 Experiments

6.4.1 Experimental setup

pre-trained Network We assume that we have gathered prior knowledge on unrelated visual tasks. This knowledge is modeled by a deep convolutional network, trained on a large-scale dataset. In our experiments, we choose to use a ResNet-18 [He+16] pre-trained on the Places365-Standard subset of Places365 [Zho+17]. We refer to this subset as Places. This subset contains around 1.8 million images across 365 classes.

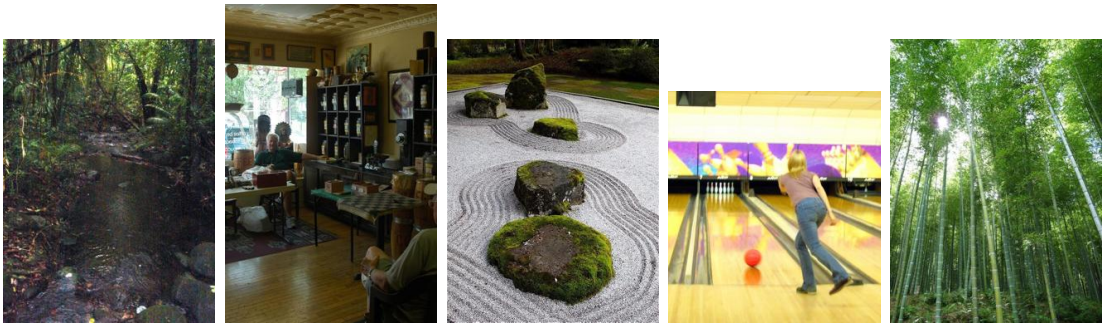


Figure 6.2 – Examples of images from the Places dataset, from left to right: rainforest, coffee shop, zen garden, bowling alley, bamboo forest.

The classes are outdoor and indoor scenes, some examples are shown in Figure 6.2. We select this dataset for its large scale, diversity of content and different nature than other popular datasets like CUB. Images are resampled to 224×224 pixels for training. We choose ResNet-18 as the architecture of the pre-trained model as it is a powerful network that is also used in other few-shot learning studies [Che+19; DSM19], which helps in comparisons. We make no assumption on the pre-training of the network. We do not access either the pre-training process or the dataset. We rather use a publicly available converged model that has been trained with a fully-connected layer as a classifier, as assumed in section 6.2.

Datasets We evaluate our method on CUB with images of resolution 224×224. We also experiment with *mini*ImageNet. Similarily to [Che+19; DSM19], we resample the *mini*ImageNet images to 224×224, which is consistent with the choice of pre-trained network.

Dataset overlap Contrary to CUB, *mini*ImageNet has some non-negligible overlap with Places. Some classes or even objects appear in both datasets. To better satisfy our assumption of domain gap, we remove the most problematic overlapping classes from *mini*ImageNet: 3 base classes, 1 validation class and 2 novel classes. We refer to this pruned dataset as *modified* mini*ImageNet*. For the sake of comparison and because this overlap can happen in practice, we also experiment on the original *mini*ImageNet.

In particular, we measure, for each *mini*ImageNet class, what is the most frequent prediction among Places classes by the pre-trained ResNet-18 classifier. We distinguish two kinds of overlap: *full overlap* when we find two classes with the same semantic definition, and *partial overlap* when a large portion of the two classes are in common.

Split	Match rate	*mini*ImageNet class	Places class	FO	PO
Train	0.89	carousel	carrousel	✓	
	0.75	rock beauty	underwater - ocean deep		
	0.68	slot	amusement arcade		✓
	0.66	yawl	boat deck		
	0.61	jellyfish	aquarium		
	0.56	dugong	underwater - ocean deep		
	0.54	aircraft carrier	landing deck		
	0.49	cliff	cliff	✓	
	0.47	triceratops	natural history museum		
	0.43	three-toed sloth	rainforest		
Validation	0.80	coral reef	underwater - ocean deep		✓
	0.51	catamaran	boat deck		
	0.21	garbage truck	loading dock		
	0.20	poncho	clothing store		
Test	0.52	school bus	bus station - indoor		✓
	0.48	bookshop	bookstore	✓	
	0.41	black-footed ferret	veterinarians office		
	0.40	theater curtain	movie theater - indoor		
	0.36	african hunting dog	watering hole		

Table 6.1 – Top ranking *mini*ImageNet classes in terms of classification to a a specific Places class and our decision on if w consider it as overlapping with the corresponding Places class. FO and PO respectively stand for full and partial overlap.

Partial overlap can occur for multimodal classes, for instance the *slot machine* class is a subclass of *amusement arcade*. Alternatively, even if the semantic definitions of the two classes are different, it can happen that the same objects are depicted in the two classes as is the case for the *school bus* and *bus station-indoor* classes. Top-ranking classes in terms of classification to a specific Places class and our decision on the overlap are shown in Table 6.1. We remove all full and partial overlapping classes from *mini*ImageNet. Table 6.2 lists the classes removed from *mini*ImageNet in this way.

Evaluation protocol To adapt to our few-shot version of few-shot learning, we randomly keep only k' images per base class. We experiment with $k' \in \{0, 1, 5, 10\}$ and $k' \in \{0, 20, 50\}$ respectively for the CUB and *mini*ImageNet. This is because CUB classes refer to bird species, while miniImageNet classes to broad object categories, hence have a lot more variability. For novel classes, we use the standard setting $k \in \{1, 5\}$. We

*mini*ImageNet split	*mini*ImageNet class	Places class with overlap
TRAIN	carousel	carousel
TRAIN	slot	amusement arcade
TRAIN	cliff	cliff
VALIDATION	coral reef	underwater - ocean deep
TEST	school bus	bus station - indoor
TEST	bookshop	bookstore

Table 6.2 – Classes removed from *mini*ImageNet to form the *modified* mini*ImageNet* dataset and the corresponding overlapping Places classes.

generate a few-shot task on novel classes by selecting a support set X'. In particular, we sample c' classes from the validation or test set and from each class we sample k images. In all experiments, $c' = 5$, *i.e.* 5-way classification. For each task we additionally sample 30 novel class images per class, to use as queries for evaluation. We report *average accuracy* and 95% *confidence interval* over 5,000 tasks for each experiment. The base class training set X contains k examples per base class in C. Each experiment can be seen as a few-shot classification task on few base class examples.

Baselines We evaluate experiments with the network being either pre-trained on Places or randomly initialized. In both cases, we report measurements for different numbers k of examples per base class, as well as *all* examples in X. In the latter case (randomly initialized), we do not use the option $k' = 0$ because then there would be no reasonable representation to adapt or to perform inference on, given a few-shot task on novel classes. Furthermore, in this case we do not apply the attention mechanism as it is based on the pre-trained classifier. In all cases, we compare to the baselines of using no adaptation and no spatial attention. When learning from scratch, spatial attention is not applied as we do not have access to the pre-trained classifier. In the case of random initialization, and using all examples in X, we compare to Baseline++ [Che+19] and prototypical networks [SSZ17], as reported in the benchmark by Chen *et al.* [Che+19], as well as *category traversal* (CTM) [Li+19a] and *ensembles* [DSM19], all using ResNet-18. They can only be compared to our randomly initialized baseline when using base training on all data.

Implementation details At base training, we use stochastic gradient descent with Nesterov momentum with mini-batches of size 200. At adaptation, we perform a maxi-

Attention		✓		✓
Adaptation			✓	✓
BASE		PLACES		
$k' = 0$	38.80±0.24	39.69±0.24	39.76±0.24	40.79±0.24
$k' = 1$	40.50±0.23	41.74±0.24	41.11±0.24	42.23±0.24
$k' = 5$	56.47±0.28	57.16±0.29	56.69±0.29	57.32±0.29
$k' = 10$	62.83±0.30	64.32±0.30	62.97±0.30	64.41±0.30
ALL	80.68±0.27	80.48±0.27	80.68±0.27	80.56±0.27
BASE		RANDOMLY INITIALIZED		
$k' = 1$	31.65±0.19	-	31.37±0.19	-
$k' = 5$	40.52±0.25	-	40.50±0.26	-
$k' = 10$	48.25±0.28	-	48.61±0.29	-
ALL	71.78±0.30	-	71.77±0.30	-
Baseline++	67.02±0.90	-	-	-
ProtoNet	71.88±0.91	-	-	-
Ensemble	68.77±0.71	-	-	-

Table 6.3 – *Average 5-way 1-shot novel class accuracy on CUB.* We use ResNet-18 either pre-trained on Places or we train it from scratch on k base class examples. ProtoNet [SSZ17] is as reported by Chen *et al.* [Che+19]. For ensemble [DSM19], we report the distilled model from an ensemble of 20. Baselines as reported in the literature, without attention or adaptation; to be compared only to randomly initialized with $k' =$ ALL.

mum of 60 iterations over the support examples using Adam optimizer with fixed learning rate. In both cases, the learning rate, schedule if any and number of iterations are determined on the validation set. The temperature used by (6.1) for the computation of the entropy of ESA is fixed per dataset, again on the validation set. In particular, we use $T = 100$ and $T = 2.6$ respectively for CUB and modified *mini*ImageNet. For the spatial attention threshold of NSA, we use $\tau = 0.7$ and $\tau = 0.6$ respectively for CUB and modified *mini*ImageNet.

6.4.2 Results

We present results in Tables 6.3 and 6.4 for CUB 1-shot and 5-shot respectively, and in Tables 6.5 and 6.6 for modified *mini*ImageNet 1-shot and 5-shot. The original *mini*ImageNet is discussed separately at the end of this section.

Attention		✓		✓
Adaptation			✓	✓
BASE		PLACES		
$k' = 0$	55.09±0.24	56.95±0.23	63.29±0.24	64.27±0.23
$k' = 1$	57.25±0.22	58.89±0.23	65.42±0.23	66.78±0.23
$k' = 5$	74.27±0.23	74.95±0.23	75.82±0.23	76.32±0.23
$k' = 10$	78.89±0.22	80.08±0.21	80.56±0.22	81.53±0.21
ALL	90.38±0.16	90.33±0.16	91.22±0.15	91.17±0.15
BASE		RANDOMLY INITIALIZED		
$k' = 1$	39.45±0.20	-	42.70±0.21	-
$k' = 5$	52.94±0.25	-	53.45±0.25	-
$k' = 10$	63.37±0.26	-	64.52±0.26	-
ALL	85.60±0.18	-	85.96±0.19	-
Baseline++	83.58±0.54	-	-	-
ProtoNet	87.42±0.48	-	-	-
Ensemble	84.62±0.44	-	-	-

Table 6.4 – *Average 5-way 5-shot novel class accuracy on CUB.* We use ResNet-18 either pre-trained on Places or we train it from scratch on k base class examples. ProtoNet [SSZ17] is as reported by Chen *et al.* [Che+19]. For ensemble [DSM19], we report the distilled model from an ensemble of 20. Baselines as reported in the literature, without attention or adaptation; to be compared only to randomly initialized with $k' =$ ALL.

Attention		✓		✓
Adaptation			✓	✓
BASE		PLACES		
$k' = 0$	61.66±0.30	63.36±0.29	62.09±0.30	63.56±0.30
$k' = 20$	62.95±0.29	63.15±0.28	63.11±0.29	63.33±0.29
$k' = 50$	65.07±0.29	65.10±0.29	65.18±0.29	65.24±0.29
ALL	66.20±0.29	65.94±0.29	66.23±0.29	66.06±0.29
BASE		RANDOMLY INITIALIZED		
$k' = 20$	33.43±0.21	-	33.35±0.21	-
$k' = 50$	41.03±0.24	-	41.05±0.24	-
ALL	55.99±0.28	-	56.13±0.28	-

Table 6.5 – *Average 5-way 1-shot novel class accuracy on modified* mini*ImageNet.* We use ResNet-18 either pre-trained on Places or we train it from scratch on k' base class examples. Baselines only shown in Table 6.7 on the original *mini*ImageNet.

Attention		✓		✓
Adaptation			✓	✓
BASE		PLACES		
$k' = 0$	78.86±0.22	80.15±0.22	80.38±0.22	81.05±0.22
$k' = 20$	78.41±0.21	78.53±0.21	79.67±0.21	79.82±0.21
$k' = 50$	79.94±0.20	79.99±0.20	80.88±0.20	80.96±0.20
ALL	80.37±0.21	80.24±0.21	81.56±0.20	81.50±0.20
BASE		RANDOMLY INITIALIZED		
$k' = 20$	43.83±0.21	-	44.21±0.21	-
$k' = 50$	54.68±0.22	-	54.92±0.22	-
ALL	72.43±0.22	-	73.10±0.21	-

Table 6.6 – *Average 5-way 5-shot novel class accuracy on modified* mini*ImageNet.* We use ResNet-18 either pre-trained on Places or we train it from scratch on k' base class examples. Baselines only shown in Table 6.8 on the original *mini*ImageNet.

Effect of base training For fine-grained few-shot classification (CUB), base training is extremely important in adapting to the new domain, improving the baseline 1-shot accuracy by more than 40% with no adaptation and no spatial attention. For general object classification (modified *mini*ImageNet), it is less important, improving by 4.5%. It is the first time that experiments are conducted on just a small subset of the base class training set. It is interesting that 50 examples per class are bringing nearly the same improvement as all examples, *i.e.* hundreds per class.

Effect of (novel class) adaptation Fine-tuning the network on k novel class examples per class, even fewer than k' in the case of base classes, comes with the risk of over-fitting. We still show that a small further improvement is possible with a small learning rate. The improvement is more significant when k is low, in which case, more adaptation of the embedding network to the novel class domain is needed. In the extreme case of CUB dataset without base training, adaptating on only the 25 images of the 5-way 5-shot tasks brings an improvement of 8.20%.

Effect of spatial attention Spatial attention allows focusing on the most discriminative parts of the input, which is more beneficial when fewer examples are available. The extreme case is having no base class images and only one image per novel class. In this case, most improvement comes on modified *mini*ImageNet without base training, where spatial attention improves 5-way 5-shot classification accuracy by 1.5% after adaptation.

Attention		✓		✓
Adaptation			✓	✓
BASE	PLACES			
$k' = 0$	65.80±0.31	67.56±0.31	66.41±0.32	67.96±0.31
$k' = 20$	66.98±0.29	67.63±0.29	67.32±0.29	67.80±0.29
$k' = 50$	69.11±0.29	69.17±0.29	69.22±0.29	69.30±0.29
ALL	69.71±0.29	69.81±0.29	69.70±0.29	70.00±0.29
BASE	RANDOMLY INITIALIZED			
$k' = 20$	37.75±0.23	-	37.74±0.23	-
$k' = 50$	42.79±0.23	-	42.79±0.23	-
ALL	59.68±0.27	-	59.66±0.27	-
Baseline++	51.87±0.77	-	-	-
ProtoNet	54.16±0.82	-	-	-
Ensemble	63.06±0.63	-	-	-
CTM	64.12±0.55	-	-	-

Table 6.7 – *Average 5-way 1-shot novel class accuracy on original* mini*ImageNet.* We use ResNet-18 either pre-trained on Places or we train it from scratch on k base class examples. ProtoNet [SSZ17] is as reported by Chen *et al.* [Che+19]. CTM refers to the data-augmented version of Li *et al.* [Li+19a]. For ensemble [DSM19], we use the distilled model from an ensemble of 20. Baselines as reported in the literature, without attention or adaptation; to be compared only to randomly initialized with $k' =$ ALL.

The attention maps appear to be domain-independent as they improve CUB accuracy even when no images from the bird domain have been seen ($k = 0$).

Original *mini*ImageNet results The original *mini*ImageNet dataset partially overlaps Places. We use $T = 2.4$ for the temperature in (6.1). The remaining setup is as for modified *mini*ImageNet. Results are shown in tables 6.7 ands 6.7.

Compared to the results of modified *mini*ImageNet (tables 6.5 and 6.6), performances are nearly uniformly increased by 3-4% and conclusions remain the same. The increase in performance is due to having more training data, as well as putting back easily classified classes in the test dataset. Observe that, unlike CUB (tables 6.3 and 6.4), CTM [Li+19a] and ensembles [DSM19] perform better than our randomly initialized baseline.

Attention		✓		✓
Adaptation			✓	✓
BASE	PLACES			
$k' = 0$	81.90±0.23	83.00±0.22	83.45±0.22	84.09±0.22
$k' = 20$	81.44±0.21	81.82±0.21	82.56±0.21	82.92±0.21
$k' = 50$	83.14±0.20	83.25±0.20	83.97±0.19	84.10±0.19
ALL	83.31±0.19	83.25±0.19	84.20±0.19	84.24±0.19
BASE	RANDOMLY INITIALIZED			
$k' = 20$	49.13±0.23	-	49.67±0.23	-
$k' = 50$	57.18±0.23	-	57.68±0.23	-
ALL	75.42±0.20	-	75.95±0.20	-
Baseline++	75.68±0.63	-	-	-
ProtoNet	73.68±0.65	-	-	-
Ensemble	80.63±0.43	-	-	-
CTM	80.51±0.13	-	-	-

Table 6.8 – *Average 5-way 5-shot novel class accuracy on original* mini*ImageNet.* We use ResNet-18 either pre-trained on Places or we train it from scratch on k base class examples. ProtoNet [SSZ17] is as reported by Chen *et al.* [Che+19]. CTM refers to the data-augmented version of Li *et al.* [Li+19a]. For ensemble [DSM19], we use the distilled model from an ensemble of 20. Baselines as reported in the literature, without attention or adaptation; to be compared only to randomly initialized with $k' =$ ALL.

	1-SHOT			5-SHOT		
BASE	Places	Places+CUB	CUB	Places	Places+CUB	CUB
No att	38.80±0.24	80.68±0.27	71.78±0.30	55.09±0.24	90.38±0.16	85.60±0.18
NSA	40.72±0.24	79.77±0.28	69.91±0.11	59.08±0.24	90.03±0.16	83.86±0.19
ESA	39.69±0.24	80.48±0.27	-	56.95±0.23	90.33±0.16	-
NSA+ESA	40.71±0.24	79.77±0.28	-	59.06±0.24	90.03±0.16	-

Table 6.9 – *Average 5-way novel class accuracy on CUB.* We use ResNet-18 either pre-trained on Places, pre-trained on Places then fine-tuned on CUB or trained from scratch on CUB using all base class examples. We are applying different spatial attention strategies at inference. ESA cannot be used when no pre-trained classifier is available.

	1-shot			5-shot		
Base	Places	Places+MMI	MMI	Places	Places+MMI	MMI
No att	61.66±0.30	66.20±0.29	55.99±0.28	78.86±0.22	80.37±0.21	72.43±0.22
NSA	63.23±0.29	65.30±0.29	53.54±0.27	80.32±0.21	79.88±0.21	70.01±0.22
ESA	63.36±0.29	65.94±0.29	-	80.15±0.22	80.24±0.21	-
NSA+ESA	63.45±0.29	65.32±0.29	-	80.43±0.21	79.87±0.21	-

Table 6.10 – *Average 5-way novel class accuracy on modified* mini*ImageNet.* We use ResNet-18 either pre-trained on Places, pre-trained on Places then fine-tuned on modified *mini*ImageNet or trained from scratch on modified *mini*ImageNet using all base class examples. We are applying different spatial attention strategies at inference. ESA cannot be used when no pre-trained classifier is available.

6.5 Comparison of spatial attention mechanisms

In Tables 6.9 and 6.10 we present the accuracy test results of the two spatial attention mechanisms, NSA and ESA, respectively on CUB and modified *mini*ImageNet. In this set of experiments, no novel class adaptation is performed. We apply spatial attention at few-shot inference only. Each attention mechanism and combination of the two is tested using embedding networks that are either just pre-trained on Places, pre-trained on Places and then fine-tuned on the base classes of the few-shot learning dataset, or trained from scratch on the few-shot dataset base classes (standard few-shot learning setup). The threshold τ of NSA is kept fixed for all experiments of a given dataset, fixed using a validation set with a pre-trained network on Places embedding network. The temperature T of ESA is kept fixed for all experiments of a given dataset except when combined with NSA, in which case it is selected again on the validation set using a pre-trained classifier as embedding network.

We observe that NSA improves few-shot accuracy on both datasets while using the network pre-trained on Places, in the case of CUB, the improvements are larger than the ones observed with ESA, they are equivalent on modified *mini*ImageNet. However, in those experiments, NSA fails in all instances of the network being trained on the full base set, decreasing the accuracy by about 2%. Note that it does not invalidate NSA completely: NSA leads to the best 5-way 1-shot and 5-shot accuracies when using the pre-trained network alone on the CUB dataset. It rather reveals that the threshold τ of NSA depends on the number of base class examples. ESA shows consistent improvements in accuracy on both datasets using the pre-trained network. When ESA is used with the same temperature T using the pre-trained and fine-tuned network, the resulting

accuracy is marginally inferior to not using it. This makes ESA a safe choice of spatial attention, as it has no effect in the worst case. We also observe that using ESA on top of NSA does not influence the few-shot accuracy in any of the tested setups. This result can be explained as in essence those two mechanisms aims at discarding the background related features of novel examples; when those are suppressed by NSA, the effect of ESA is very limited.

6.6 Conclusion

In this chapter we generalized the problem of few-shot learning by studying the case where even base classes images are limited in number, introducing a new setup called few-shot few-shot learning. To address it, we use a pre-trained network on a large-scale dataset and a very simple spatial attention mechanism that does not require any training on the base or novel classes. This spatial attention (ESA) is based on the class prediction of the pre-trained network and improves few-shot classification accuracy in case where base class examples are lacking and does not damage it when base class images are available. We consider two few-shot learning datasets: CUB and *mini*ImageNet, with different domain gaps to our prior dataset Places. Our findings indicate that even when the domain gap is large between the dataset used for pre-training and the base/novel class domains, it is still possible to get significant benefit from base class training even with a few examples, which is very important as it reduces the need for in-domain supervision.

Chapter 7

Application to classification of aerial images

Contents

In all previous chapters, we explored the issues linked to few-shot learning and introduced models to address those. This exploration aims at improving the state of the art knowledge in this domain. In chapter 3, chapter 4, we worked on the standard few-shot learning setting, used in most few-shot classification works. In chapter 5 we depart from the standard few-shot learning setup to work on transductive few-shot learning which is now also a commonly studied task. In chapter 6 we propose a novel, more realistic few-shot learning setting called few-shot few-shot learning. This new setup is still general enough to be picked up by the research community. In all those cases, we applied our analysis and models to commonly used datasets: *mini*ImageNet, CIFAR100 and CUB. The first two can be considered as general classification. The last one is in a specific domain, bird species, and is used as example of fine-grained classification. When using

CUB, we always also used a general classification dataset to show the generalization to other domains to guarantee the behavior of our models in the general case.

In this chapter, we depart from this protocol to focus on a specific application of few-shot classification. In particular, we present a problem of interest at Safran. More specifically, the objective is to classify land vehicles in aerial images. The task introduces new challenges as the classes are few even for the base training set, the images are small and background is treated as a separate class. This chapter aims at giving insights on the application of previous few-shot learning methods on similar practical tasks.

7.1 Problem

In this section, we introduce the problem addressed in this section. Detailing the data and constraints.

7.1.1 Data

For the sake of confidentiality, images from the dataset used for this task cannot be released. Through an internal data collection campaign, people at Safran gathered a number of large aerial images. Contrary to the other datasets used in this thesis where images were RGB, the images here are infrared, encoded as single channel grayscale images. Examples of aerial images similar to those used for this study is shown in Figure 7.1. Because the object of interest in those images are vehicles, a detection method was first used, in particular Faster RER-CNN [TJ18]. This model is based on Faster R-CNN [Ren+15] which is composed of a detector network that proposes regions of interest and a classifier that classifies the corresponding region. The regions are rectangular and axis-aligned. In Faster RER-CNN, similarily to [Din+19], the region proposal network proposes regions of interest with an orientation. This fits the task at hand where vehicles are found in images without any preferred orientation.

With this method, 8051 regions of interest with highest detection scores are selected. Corresponding image patches are extracted. The resulting images have small resolution (about 30 pixels wide and long). For simplicity all images are resized to squared images of 85×85 pixels. The images were warped to all have the same orientation, which we found to help on the classification task. Because the detection model detects some false positives, some images correspond to background with any vehicle.

Those images were then manually annotated into specialized classes, being the type of vehicle or background. More specifically, we regroup most vehicles into a common

Figure 7.1 – Examples of images from the VEDAI dataset [RJ15]. Uncropped images used in this study are similar in size and content.

Meta-class	Number of images
Generic vehicles	4405
Background	3581
Rare vehicles	65

Table 7.1 – Detailed repartition of vehicle images in the Safran dataset.

meta-class, called generic vehicles. We are most interested in rarely seen vehicle types, which will form our novel classes. In Table 7.1, we show the number of images from each of those meta classes used in this study.

7.1.2 Task

Many different tasks could be considered using the data presented above. In this work, we focus on the classification of rare vehicles in infrared aerial images. The choice was made to focus on developing a multiclass classifier that would classify into three classes: background, generic vehicles and one of the rare vehicles class. Ultimately, it would be interesting to be able to add all other rare vehicle classes to the classification. This task would be similar to the few-shot classification into both base classes and novel classes as done in [QBL18; GK18] and in this thesis in chapter 3. Still, we chose not go in this direction because of the data limitations.

Inside the rare vehicles meta-class, most vehicle classes have very few examples. In Figure 7.2 we show the distribution of the number of images of the 4 largest rare vehicles classes. This work being a first approach to treating those classes, we chose to limit the study to the largest two classes of rare vehicles having respectively 18 and 15 examples.

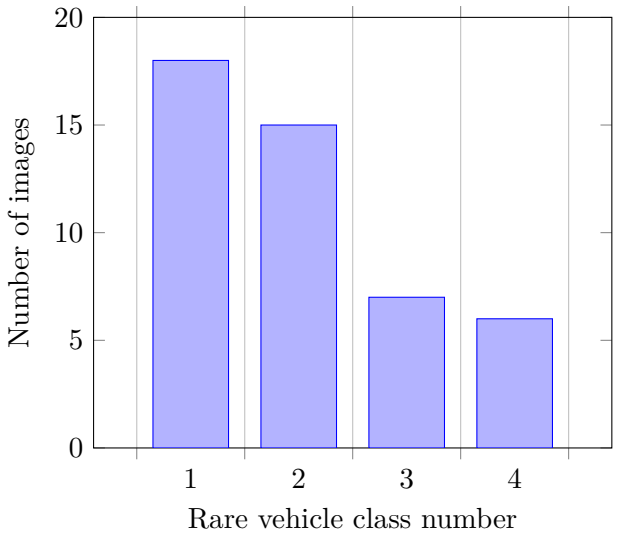


Figure 7.2 – Number of images for the four largest rare vehicles classes.

Other classes all have seven or less examples. Performance using such extremely scarce data would be heavily dependent on the choice of the quality of the examples. In a work that aims at drawing conclusions on a new task, those classes might introduce an unwanted bias.

Using the standard few-shot learning setting was not satisfactory. Indeed, few-shot learning datasets usually have many distinct classes fully dedicated to validation or testing. In our case, we cannot afford to have novel validation classes, therefore we keep some images in each class for this purpose and validate using the same novel classes. Because of this, we cannot guarantee generalization to other novel classes but we can at least verify consistency between the two used classes. Additionally, if we were to use both rare vehicle classes together in classification, it would not guarantee a good performance with other combinations of rare vehicle classes. With other few-shot datasets, one can artificially generate thousands of different few-shot tasks by picking a subset of the novel classes available which is not possible here.

7.2 Model

In this section are presented the choices made for the model used for the aerial images classification task. The task here is not exactly defined as previous chapters few-shot tasks as we are limited to only two classes for base classes and only one novel class. Nevertheless, we adopt a similar method framework as previous chapters. Specifically,

first we use base classes to learn a representation of images. Then we use few-shot data to build a classifier for the novel class.

7.2.1 Experimental setup

In previous chapters, testing a method was performed by applying it on a high number of few-shot tasks sampled from the validation or test dataset. This task sampling involved sampling a subset of novel classes, then inside those classes, a subset of images to be used as support examples and a disjoint subset to be used as query examples. In this study, we only have one novel class in the classification. Therefore, the class sampling step is skipped. We still sample a large number of tasks by sampling different combinations of supports and query examples in the novel class. We report the average results separately for the two possible novel classes.

7.2.2 Base class learning

Choice of embedding network We experimented with two networks for the embedding function: ResNet-18 and ResNet-12. Without pretraining ResNet-12 proved to be a better choice in other dataset experiments. Because ResNet-18 is a more common architecture, we could find and use a version of it pre-trained on *mini*ImageNet. Consistent with what was observed in chapter 6, even though there is a large domain gap between *mini*ImageNet and aerial images, this pre-trained network resulting in the best overall results.

Choice of training loss We used the same framework as described in chapter 3, that is to say train the embedding network on a classification task on the base classes. In this particular task, the base classes are only two: background and generic vehicles. Therefore, the classification task becomes a binary classification task. We experimented with two losses for binary classification. On the one hand, we used the usual multi-class cross-entropy as in chapter 3, that is, using a one layer classifier parametrized by two weight vectors for the two classes. In this case, we enforce background images to be mapped to a particular point in the feature space. On the other hand, we modified the classification layer to only have one output and applied a binary cross-entropy loss on it. In this case, we enforce background images to be mapped far away from the vehicle images. In this case (2.15) becomes

$$f_{\theta,W}(\mathbf{x}) := S\left(s_\tau(\phi_\theta(\mathbf{x}), W)\right) \tag{7.1}$$

	Task RV1		Task RV2	
	Recall novel	Accuracy	Recall novel	Accuracy
Loss	Binary cross-entropy			
1-shot	0.28	0.66	0.23	0.64
5-shot	0.70	0.82	0.42	0.72
10-shot	0.81	0.86	0.63	0.79
Loss	Multiclass cross-entropy			
1-shot	0.04	0.62	0.1	0.71
5-shot	0.42	0.74	0.11	0.61
10-shot	0.71	0.83	0.41	0.71

Table 7.2 – Average recall of the novel class and accuracy with multiclass cross-entropy and binary cross-entropy loss. The accuracy is computed using both base classes and novel classes. Dense classification is used in both cases. RV1 and RV2 refers to rare vehicle classes 1 and 2.

where $W \in \mathbb{R}^d$ is the only parameter vector associated with the positive class and S is the sigmoid function

$$S(x) := \frac{1}{1+e^{-x}} \tag{7.2}$$

with $x \in \mathbb{R}$. Similarly to dense classification, equation 3.1 becomes

$$f_{\theta,W}(\mathbf{x}) := \left[S\left(s_\tau(\phi_\theta^{(k)}(\mathbf{x}), W)\right)\right]_{k=1}^{r} \tag{7.3}$$

The result loss function is

$$\ell(p, y) := -y \log p - (1-y)\log(1-p) \tag{7.4}$$

with $p \in \mathbb{R}$. The resulting loss function minimization aims at enforcing that positive examples representations are grouped together, close to W, while having the representation of the negative examples far from it. The generic vehicle class is the natural choice for positive class. Because we are mostly interested in the rare vehicle class performance, we measure the recall for the rare vehicle class, that is to say the proportion of rare vehicle images that are classified properly at test time. We also measure the average accuracy as done in previous chapters. Table 7.2 shows the average recall of the novel class and total accuracy using those two losses for the representation learning stage. For both rare vehicle class tasks and for all measures, we observe superior performance

	Task RV1		Task RV2	
	Recall novel	Accuracy	Recall novel	Accuracy
Loss	Dense classification			
1-shot	0.28	0.66	0.23	0.64
5-shot	0.70	0.82	0.52	0.72
10-shot	0.81	0.86	0.63	0.79
Loss	Global average pooling			
1-shot	0.41	0.71	0.30	0.66
5-shot	0.61	0.76	0.51	0.70
10-shot	0.74	0.82	0.54	0.72

Table 7.3 – Average recall of the novel class and accuracy with global average pooling and dense classification. The accuracy is computed using both base classes and novel classes. Binary cross-entropy is used in both cases. RV1 and RV2 refers to rare vehicle classes 1 and 2.

using the binary cross-entropy loss. Similarly in Table 7.3 we compare performances on the novel class while using dense classification or regular global pooling. We observe superior performance using dense classification for 5-shot and 10-shot for both classes. Average pooling gives better performance in the 1-shot scenario.

7.2.3 Novel class adaptation

Novel classes classifier Similar to what was done in subsection 4.1.4, we build a new one layer classifier by computing prototypes. There are however two differences. Firstly, in this case we only have one novel class (one rare vehicle for each few shot task). Secondly, we also produce prototypes for the base classes. In chapter 3, we reused the learned parameters W for representative for the base classes. Here, because we used a binary classifier, $W \in \mathbb{R}^d$ is a unique representative and therefore cannot be used for both base classes. Prototypes for the base classes only have to be computed once at the end of the representation learning stage, the computational cost is limited to a single forward pass of the training dataset which is minimal. We also experimented with using W as the prototype of the generic vehicle class and computing only a prototype for the background class which gave worse results.

Novel class adaptation We attempted to adapt the representation of the embedding network with few steps adaptation as done in section 4.3. As observed in this chapter,

adaptation helps for some selection of support examples and fails for others. In section 4.3 the early stopping was based on observing the number of training iterations that on average improve classification on the validation classes. In this context, we do not have the necessary data to implement this solution safely. Moreover, the observed improvements were minimal. In the end, we did not choose to use novel class adaptation.

Spatial attention Following the work done on spatial attention in subsection 6.3.2, we experimented with spatial attention for the aerial image classification task. Examination of the feature maps as done in Figure 5.2 confirms that relevant regions of the images have local descriptors of higher norms than non-discriminative regions. However, we did not observe significant performance change by using the spatial attention. A possible explanation is that the objects of interest in the images already cover most of the image surface, so there are fewer distractor background regions to eliminate. Moreover, one of the classes in this task is the background itself. This has two consequences on the model. Firstly, because the background class is part of the base classes, the embedding network might have learned to embed background regions with high norm descriptors. Secondly, any spatial attention method that aims at discarding the background features or their influence in the classification would not make sense when some of the images are purely background. For this task and any tasks that include classification into a background class, another method should be designed. A possible direction would be to exclude the class from the classifier and instead classify images where few area are judged discriminative as background. Due to lack of time, such study is left for future work.

7.3 Conclusion

In this chapter, we explored one possible application of few-shot learning methods. This work is preliminary but allows us to confirm that our observations on standard academic settings and datasets can transfer to the aerial image classification task. Dense classification shows improvements on classifying the novel queries in most cases. The norm can be used as a viable indicator that a region is relevant for classification. However method used for the standard few-shot learning settings cannot be directly applied to this problem. The task presented here has many specificities that would require more exploration. The fact the the base classes are limited to two is different from the usual many base class setup. We touched on having a small base dataset in chapter 6, however we limited the number of examples per class and not the number of classes. Moreover

one of the class is the background class. Treating background samples as negative examples in a binary classification loss is a way of dealing with this, but other options could be explored in relation to spatial attention.

Conclusion and perspectives

Representation learning In this thesis, we examined the problem of few-shot classification through the focus of representation learning and using the few support examples to adapt as much as possible to the few-shot task. First, in chapter 3, we discussed how to properly use a disjoint set of images, the base dataset, to learn a task-independent embedding of the images. We experimented with models that implement the embedding function with convolutional neural networks. We showed that the choice of CNN architecture should not be overlooked, in particular the tested residual networks show impressive results while being fast to train. With such powerful architectures, the simple cosine classifier strategy consisting in training on a multiclass classification task on the base classes results in strong few-shot performance, exceeding that of more complex meta-learning methods.

Since representation learning is used to learn an embedding that ultimately will be used to classify novel classes, it is important that it learns to represent as many discriminative details in the images as possible. With this in mind, we proposed to modify the way spatial information is treated by the neural network during representation learning. Convolution layers of CNNs return tensors with spatial dimensions. By selecting a specific pixel, we can extract a local representation of a region of the image. Usually, spatial information is suppressed by the application of flattening or spatial pooling on the output of the embedding network. Instead, in *dense classification*, we treat each local representation independently. During training, all spatial locations must be mapped by the cosine classifier to the label of the image. This simple modification shows impressive improvements for classification into the novel classes, as well as classification on the union of base and novel classes.

Adaptation In the first stage described above, the goal is to learn a task-invariant embedding function, as we do not have access to any information on the few-shot task. In chapter chapter 4, we explored methods that adapt it to be task-dependent, condi-

tioned on the novel classes' support examples. This stage is challenging as the lack of data makes any learning-based strategy prone to overfitting. We introduced two solutions. First, *implanting* a limited number of new parameters to the embedding network. Implants are convolutional layers that operate in parallel to the original embedding network. Only the implant parameters are trained on the support examples, effectively limiting the adaptation to not depart too much from the original embedding function. We also proposed an alternative strategy consisting in *fine-tuning* entire blocks of the embedding network for a small number of parameter updates. The number of steps is based on the observed optimal accuracy on a set of validation tasks. Such early stopping strategy, while being simple, allows to adapt the representation without overfitting. Those methods show limited improvement for one-shot learning. In this case, we showed that it is possible to *augment* the support set with related images from the base dataset to artificial create more shots for the task.

Role of data Continuing on the idea of utilizing the few available data as much as possible, we also studied in chapter 5 the task of *transductive* few-shot learning. In this task, the queries to classify are multiple. We can consider them as extra unlabelled data that add further information about the novel classes. Organizing examples as nodes in a graph allows using propagation methods to label all of them, including the queries. To explore local similarities in images, which can help to classify them, we proposed *local propagation*, whereby we use local representations of images as nodes in the graph. In this context, we also introduced a simple spatial attention mechanism that selects only the non-background regions for the graph. This method shows competitive results in the standard few-shot learning setting and can efficiently take advantage of unlabeled data in the transductive setting, making it a safe choice for few-shot classification in general.

Another source of data that is not considered in the standard few-shot learning setting is all the large-scale datasets that are accessible. Usually, the representation stage is limited to learning an embedding from scratch with the base class dataset. In some cases, such base dataset is not available or limited to a few samples. Based on those observations, we proposed, in chapter 6, *few-shot few-shot* learning as a novel few-shot learning setting where prior knowledge is modeled as a large-scale dataset or directly a CNN trained on such a dataset. Representation learning with base classes becomes adaptation to the few-shot task domain, which can be performed similarly as before. In this context, we introduced another spatial attention mechanism based on utilizing predictions from the pre-trained classifier. We studied the effect of the spatial attention mechanism and of previously introduced methods in this setting with varying sizes of

the base dataset. We showed that in all cases, pretraining on the large-scale dataset improves the few-shot performance, even when its domain is far from the few-shot task domain. We also observed that adaptation and spatial attention can help compensate for the lack of base classes data to adapt the domain.

Aerial data Finally, in chapter 7, we focused on a particular few-shot learning problem, namely, classification of rare vehicle classes in aerial images. This is a hard task since images have small resolution and base classes are limited to two: generic vehicles and background. Preliminary experiments showed that dense classification, applied with a binary classifier, improves few-shot accuracy. The spatial attention mechanism also seems to select relevant regions in the images. However, since one class is background, it does not result in improved performances.

Perspectives

Augmentation with related base local features In parallel to [ALG19], we experimented with the idea of augmenting the support set with related images from the base classes. More precisely, for each novel class, we selected the images with the closest representation to the class prototype and added them to the support set of the class. We observed encouraging results for one-shot classification. A finer version of this method would be to select the related local representations from the base dataset to use for enriching the support set. For instance, we could stitch together local representations of different base examples to form a global one that can be associated with a novel class. We cannot expect the extra generated examples from the base dataset to carry as much information on the novel classes as the support examples. Incorporating them in the adaptation stage could be attempted by considering them as noisy data and applying training methods dealing with such data noise [Son+20]. We have explored this direction with inconclusive results, a more in-depth study would be interesting.

Integrating a distractor class In a few-shot task, we usually assume that the queries to classify belong to one of the novel classes. In practice, such assumption cannot always hold. It would be interesting for the classifier to be able to predict that the image is in none of the novel classes as done in open-set recognition [GHC20]. A possible solution would be to introduce a distractor class in the classifier. [Ren+18b] introduces such distractor but only to prevent unlabeled data to be confused with one of the novel classes, not for final prediction. Possible directions for this work would be to use spatial

attention mechanisms to detect images that are mainly background, or reusing part of the base class data as examples of external classes.

Other few-shot learning tasks While early few-shot learning methods focus on classification, few-shot is not limited to it. A related task that has gained traction recently is few-shot object detection. It consists in learning to locate and classify objects. A few methods [Che+18; Kar+19; Fan+20] propose a modified version of Faster R-CNN [Ren+15]. The classifier part of Faster R-CNN needs to solve a few-shot classification problem, so the insights developed on classification can transfer to the detection task. Moreover, the region proposal network selects parts of the images that contain an object to classify, which is related to our effort on spatial attention to select only the relevant information. Overall, the state of the art is less mature on this task, so there is more to explore.

Incremental few-shot learning Incremental learning corresponds to the setting where a model learns to solve a task and must then be able to learn new tasks when given new training data, without forgetting the tasks learn before. The most common setting is learning to classify among a set of classes, then learning new classes. In this sense, few-shot classification where we classify among the union of base and novel classes, can be seen as a particular case of incremental learning where there is only one set of new classes to learn and examples for those classes are few. Incremental learning methods also use similar training processes such as the use of cosine classifiers [Hou+19b]. The challenge of incremental learning is to make sure that previous classes are not forgotten. Methods have been proposed to store a subset of previously seen data [RKL16; Cas+18] or intermediate representations [Ahm+20]. In which case, the adaptation of the model to new classes is done with few examples of all classes. Another solution is to learn generative models to produce data of previously seen classes [KK18], which is also a common approach for few-shot learning. Additionally, forgetting is avoided by using knowledge distillation methods [LH18], which can be applied spatially [Dha+19] for improved performance. Our focus on spatial representation could be useful when studying incremental learning.

Bibliography

[Ahm+20] I. Ahmet, Z. Jeffrey, L. Svetlana, and S. Cordelia. “Memory-Efficient Incremental Learning Through Feature Adaptation”. In: *ECCV* (2020), pp. 699–715 (cit. on p. 126).

[ALG19] A. Afrasiyabi, J.-F. Lalonde, and C. Gagné. “Associative Alignment for Few-shot Image Classification”. In: *arXiv preprint arXiv:1912.05094* (2019) (cit. on pp. 25, 73, 125).

[AS19] A. Antoniou and A. J. Storkey. “Learning to Learn By Self-Critique”. In: *NIPS*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. 2019 (cit. on p. 15).

[BB08] L. Bottou and O. Bousquet. “The Tradeoffs of Large Scale Learning”. In: *NIPS*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. Curran Associates, Inc., 2008, pp. 161–168 (cit. on p. 13).

[BBC91] Y. Bengio, S. Bengio, and J. Cloutier. “Learning a synaptic learning rule”. In: *IJCNN-91-Seattle International Joint Conference on Neural Networks*. Vol. ii. 1991, p. 969 (cit. on p. 15).

[Ben+11] Y. Bengio, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver. “Deep learning of representations for unsupervised and transfer learning”. In: *in Proc. of ICML*. 2011 (cit. on pp. 96, 97).

[Ber+16] L. Bertinetto, J. F. Henriques, J. Valmadre, P. Torr, and A. Vedaldi. “Learning feed-forward one-shot learners”. In: *NIPS*. 2016 (cit. on p. 23).

[Ber+17] G. Bertasius, L. Torresani, S. X. Yu, and J. Shi. “Convolutional Random Walk Networks for Semantic Image Segmentation”. In: *CVPR*. 2017 (cit. on p. 83).

[Bro+94] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. “Signature Verification using a "Siamese" Time Delay Neural Network”. In: *NIPS*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1994, pp. 737–744 (cit. on p. 16).

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166 (cit. on pp. 35, 37).

[BSI08a] O. Boiman, E. Shechtman, and M. Irani. “In Defense of Nearest-Neighbor Based Image Classification”. In: *CVPR*. 2008 (cit. on p. 18).

[BSI08b] O. Boiman, E. Shechtman, and M. Irani. “In defense of Nearest-Neighbor based image classification”. In: *CVPR*. 2008 (cit. on p. 16).

[BU05] E. Bart and S. Ullman. “Cross-Generalization: Learning Novel Classes From a Single Example By Feature Replacement”. In: *CVPR*. 2005 (cit. on p. 80).

[Cas+18] F. M. Castro, M. J. Marin-Jimenez, N. Guil, C. Schmid, and K. Alahari. “End-to-End Incremental Learning”. In: *ECCV*. Sept. 2018 (cit. on p. 126).

[Che+18] H. Chen, Y. Wang, G. Wang, and Y. Qiao. “LSTD: A Low-Shot Transfer Detector for Object Detection”. In: *AAAI*. 2018 (cit. on p. 126).

[Che+19] W. Chen, Y. Liu, Z. Kira, Y. F. Wang, and J. Huang. “A Closer Look at Few-shot Classification”. In: *ICLR* (2019) (cit. on pp. 7, 21, 31, 86, 102, 104–106, 108, 109).

[CSZ10] O. Chapelle, B. Schlkopf, and A. Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010 (cit. on p. 78).

[Dha+19] P. Dhar, R. V. Singh, K.-C. Peng, Z. Wu, and R. Chellappa. “Learning Without Memorizing”. In: *CVPR*. June 2019 (cit. on p. 126).

[Din+19] J. Ding, N. Xue, Y. Long, G.-S. Xia, and Q. Lu. “Learning RoI Transformer for Oriented Object Detection in Aerial Images”. In: *CVPR*. June 2019 (cit. on p. 114).

[Dou+18] M. Douze, A. Szlam, B. Hariharan, and H. Jégou. “Low-Shot Learning With Large-Scale Diffusion”. In: *CVPR*. 2018 (cit. on pp. 26, 32).

[DSM19] N. Dvornik, C. Schmid, and J. Mairal. “Diversity with Cooperation: Ensemble Methods for Few-Shot Classification”. In: *ICCV* (2019) (cit. on pp. 26, 86, 102, 104–106, 108, 109).

[DT05] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *CVPR*. Vol. 1. 2005, 886–893 vol. 1 (cit. on p. 3).

[Elb+20] D. Elbrächter, D. Perekrestenko, P. Grohs, and H. Bölcskei. “Deep neural network approximation theory”. In: *IEEE Transactions on Information Theory, submitted Jan. 2019, revised* (June 2020) (cit. on p. 13).

[EMH20] T. Elsken, J. Metzen, and F. Hutter. “Neural Architecture Search: A Survey”. In: *jmlr* (2020) (cit. on p. 35).

[Eve+98] R. M. Everson, A. K. Prashanth, M. Gabbay, B. W. Knight, L. Sirovich, and E. Kaplan. “Representation of spatial frequency and orientation in the visual cortex”. In: *Proceedings of the National Academy of Sciences* 95.14 (1998), pp. 8334–8338 (cit. on p. 3).

[FAL17] C. Finn, P. Abbeel, and S. Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML*. 2017 (cit. on pp. 6, 15, 22, 23, 28, 31, 34, 35, 55, 65, 66, 96, 100).

[Fan+20] Q. Fan, W. Zhuo, C.-K. Tang, and Y.-W. Tai. "Few-Shot Object Detection With Attention-RPN and Multi-Relation Detector". In: *CVPR*. June 2020 (cit. on p. 126).

[GB04] Y. Grandvalet and Y. Bengio. "Semi-Supervised Learning by Entropy Minimization". In: *NIPS*. 2004 (cit. on p. 76).

[GB18] V. Garcia and J. Bruna. "Few-Shot Learning with Graph Neural Networks". In: *ICLR*. 2018 (cit. on pp. 19, 20, 26, 76).

[GHC20] C. Geng, S. .-.-J. Huang, and S. Chen. "Recent Advances in Open Set Recognition: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1 (cit. on p. 125).

[GHM07] A. Gidudu, G. Hulley, and T. Marwala. "Image Classification Using SVMs: One-against-One Vs One-against-All". In: (2007) (cit. on p. 4).

[Gid+19] S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord. "Boosting few-shot visual learning with self-supervision". In: *ICCV*. 2019 (cit. on pp. 7, 27, 39).

[GK18] S. Gidaris and N. Komodakis. "Dynamic Few-Shot Visual Learning without Forgetting". In: *CVPR*. 2018 (cit. on pp. 21, 22, 31, 41, 42, 46, 48, 49, 51–53, 55, 63, 65, 77, 80, 115).

[GL14] Y. Ganin and V. Lempitsky. "Unsupervised Domain Adaptation By Backpropagation". In: (2014) (cit. on pp. 96, 97).

[GMS05] M. Gori, G. Monfardini, and F. Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, 729–734 vol. 2 (cit. on p. 19).

[Goo+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". In: *NIPS*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680 (cit. on p. 25).

[Gra06] L. Grady. "Random Walks for Image Segmentation". In: *IEEE Trans. PAMI* 28.11 (2006), pp. 1768–1783 (cit. on p. 83).

[GSK18] S. Gidaris, P. Singh, and N. Komodakis. "Unsupervised Representation Learning by Predicting Image Rotations". In: *ICLR*. 2018 (cit. on p. 27).

[HA15] E. Hoffer and N. Ailon. "Deep Metric Learning Using Triplet Network". In: *SIMBAD*. 2015 (cit. on p. 58).

[Hao+19] F. Hao, F. He, J. Cheng, L. Wang, J. Cao, and D. Tao. "Collect and Select: Semantic Alignment Metric Learning for Few-Shot Learning". In: *ICCV*. 2019 (cit. on pp. 77, 80).

[He+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *CVPR*. 2016 (cit. on pp. 35, 37, 39, 101).

[HG17] B. Hariharan and R. B. Girshick. "Low-shot Visual Recognition by Shrinking and Hallucinating Features". In: *ICCV* (2017) (cit. on pp. 24, 71).

[HHS18] E. Hoffer, I. Hubara, and D. Soudry. "Fix your classifier: the marginal value of training the last weight layer". In: *ICLR* (2018) (cit. on p. 49).

[Hil+18] N. Hilliard, L. Phillips, S. Howland, A. Yankov, C. D. Corley, and N. O. Hodas. "Few-Shot Learning with Metric-Agnostic Conditional Embeddings". In: *CoRR* abs/1802.04376 (2018) (cit. on p. 28).

[Hou+19a] R. Hou, H. Chang, M. Bingpeng, S. Shan, and X. Chen. "Cross Attention Network for Few-shot Classification". In: *NeurIPS*. 2019 (cit. on pp. 77, 80).

[Hou+19b] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin. "Learning a Unified Classifier Incrementally via Rebalancing". In: *CVPR*. June 2019 (cit. on p. 126).

[HS88] C. Harris and M. Stephens. "A Combined Corner and Edge Detector". In: *Alvey Vision Conference*. 1988 (cit. on p. 3).

[HS97] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 18, 23).

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009 (cit. on p. 26).

[HVD15] G. Hinton, O. Vinyals, and J. Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015) (cit. on pp. 26, 49).

[IS15] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:-1502.03167* (2015) (cit. on pp. 27, 35, 39).

[Isc+15] A. Iscen, G. Tolias, P.-H. Gosselin, and H. Jégou. "A comparison of dense region detectors for image search and fine-grained classification". In: *IEEE Transactions on Image Processing* (2015), p. 00 (cit. on p. 3).

[Isc+17] A. Iscen, G. Tolias, Y. Avrithis, T. Furon, and O. Chum. "Efficient Diffusion on Region Manifolds: Recovering Small Objects with Compact CNN Representations". In: *CVPR*. 2017 (cit. on pp. 83, 90).

[Isc+19a] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. "Label Propagation for Deep Semi-Supervised Learning". In: *CVPR*. 2019 (cit. on p. 76).

[Isc+19b] A. Iscen, G. Tolias, Y. Avrithis, O. Chum, and C. Schmid. "Graph convolutional networks for learning with few clean and many noisy labels". In: *arXiv preprint arXiv:1910.00324* (2019) (cit. on pp. 26, 32).

[Jég+10] H. Jégou, M. Douze, C. Schmid, and P. Pérez. "Aggregating local descriptors into a compact image representation". In: *CVPR*. 2010, pp. 3304–3311 (cit. on p. 3).

[Kar+19] L. Karlinsky, J. Shtok, S. Harary, E. Schwartz, A. Aides, R. Feris, R. Giryes, and A. M. Bronstein. "RepMet: Representative-Based Metric Learning for Classification and Few-Shot Object Detection". In: *CVPR*. June 2019 (cit. on p. 126).

[KB14] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv* (2014) (cit. on pp. 52, 63).

[KH92] A. Krogh and J. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *NIPS*. 1992 (cit. on p. 65).

[Kim+18] J. Kim, Y. Choi, M. Cha, J. Lee, S. Lee, S. Kim, Y. Choi, and J. Kim. "Auto-Meta: Automated Gradient Based Meta Learner Search". In: *NIPS 2018 Workshop on Meta-Learning (MetaLearn 2018)*. Vol. abs/1806.06927. 2018 (cit. on p. 15).

[KK18] R. Kemker and C. Kanan. "FearNet: Brain-Inspired Model for Incremental Learning". In: *ICLR*. 2018. URL: `https://openreview.net/forum?id=SJ1Xmf-Rb` (cit. on p. 126).

[KLL08] T. H. Kim, K. M. Lee, and S. U. Lee. "Generative Image Segmentation Using Random Walks with Restart". In: *ECCV*. Springer. 2008, pp. 264–275 (cit. on p. 83).

[KMO16] Y. Kalantidis, C. Mellina, and S. Osindero. "Cross-dimensional weighting for aggregated deep convolutional features". In: *ECCVW*. 2016 (cit. on p. 81).

[Kra+16] P. Krahenbuhl, J. Donahue, T. Darrell, and A. Efros. "Context Encoders: Feature Learning by Inpainting". In: *CVPR*. 2016 (cit. on p. 27).

[Kri09] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009 (cit. on p. 28).

[Kuz+18] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, and T. Duerig. "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale". In: *arXiv preprint arXiv:1811.00982* (2018) (cit. on p. 96).

[KW14] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *ICLR*. 2014 (cit. on p. 25).

[KZS15] G. Koch, R. Zemel, and R. Salakhutdinov. "Siamese neural networks for one-shot image recognition". In: *ICMLW*. 2015 (cit. on pp. 16, 17, 20).

[LA17] S. Laine and T. Aila. "Temporal Ensembling for Semi-Supervised Learning". In: *ICLR* (2017) (cit. on p. 76).

[Lak+11] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. "One shot learning of simple visual concepts". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. 2011 (cit. on p. 28).

[Lee+19] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. "Meta-Learning with Differentiable Convex Optimization". In: *CVPR*. 2019 (cit. on p. 39).

[Lee13] D.-H. Lee. "Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks". In: (2013) (cit. on p. 76).

[LH18] Z. Li and D. Hoiem. "Learning Without Forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (Dec. 2018), pp. 2935–2947 (cit. on pp. 97, 126).

[LH19] I. Loshchilov and F. Hutter. "Decoupled Weight Decay Regularization". In: *ICLR* (2019) (cit. on p. 63).

[Li+19a] H. Li, D. Eigen, S. Dodge, M. Zeiler, and X. Wang. "Finding Task-Relevant Features for Few-Shot Learning by Category Traversal". In: *CVPR*. 2019 (cit. on pp. 31, 77, 80, 104, 108, 109).

[Li+19b] W. Li, L. Wang, J. Xu, J. Huo, Y. Gao, and J. Luo. "Revisiting Local Descriptor Based Image-To-Class Measure for Few-Shot Learning". In: *CVPR*. 2019 (cit. on pp. 18, 76, 80, 87, 91).

[Lia+20] D. Lian, Y. Zheng, Y. Xu, Y. Lu, L. Lin, P. Zhao, J. Huang, and S. Gao. "Towards Fast Adaptation of Neural Architectures with Meta Learning". In: *ICLR*. 2020 (cit. on p. 15).

[Liu+19a] M.-Y. Liu, X. Huang, A. Mallya, T. Karras, T. Aila, J. Lehtinen, and J. Kautz. "Few-shot unsupervised image-to-image translation". In: *ICCV*. 2019 (cit. on pp. 20, 25, 26, 71).

[Liu+19b] Y. Liu, J. Lee, M. Park, S. Kim, E. Yang, S. Hwang, and Y. Yang. "Learning to Propagate Labels: Transductive Propagation Network for Few-Shot Learning". In: *ICLR*. 2019 (cit. on pp. 76, 78, 91, 92).

[Liu+19c] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu. "Large-Scale Long-Tailed Recognition in an Open World". In: *CVPR*. 2019 (cit. on pp. 32, 87).

[Liu+20] B. Liu, Y. Cao, Y. Lin, Q. Li, Z. Zhang, M. Long, and H. Hu. "Negative Margin Matters: Understanding Margin in Few-shot Classification". In: *arXiv preprint arXiv:2003.12060* (2020) (cit. on p. 91).

[Low04] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), pp. 91–110 (cit. on p. 3).

[LSD15] J. Long, E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CVPR*. 2015 (cit. on p. 49).

[LST15] B. Lake, R. Salakhutdinov, and J. Tenenbaum. "Human-level concept learning through probabilistic program induction". English (US). In: *Science* 350.6266 (Dec. 2015), pp. 1332–1338 (cit. on p. 28).

[Ma+21] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan. "Image Matching from Handcrafted to Deep Features: A Survey". In: *International Journal of Computer Vision* 129 (Jan. 2021) (cit. on pp. xxii, 2).

[Mah+18] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. "Exploring the limits of weakly supervised pretraining". In: *ECCV*. 2018, pp. 181–196 (cit. on p. 96).

[Man+20] P. Mangla, N. Kumari, A. Sinha, M. Singh, B. Krishnamurthy, and V. N. Balasubramanian. "Charting the right manifold: Manifold mixup for few-shot learning". In: *WACV*. 2020 (cit. on pp. 7, 27, 39).

[MDL18] A. Mallya, D. Davis, and S. Lazebnik. "Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights". In: *ECCV*. 2018 (cit. on pp. 59, 97).

[Men+13] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2624–2637 (cit. on p. 4).

[Met+19] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-dickstein. "Meta-Learning Update Rules for Unsupervised Representation Learning". In: *ICLR*. 2019 (cit. on p. 15).

[Mis+18] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. "A simple neural attentive meta-learner". In: *ICLR* (2018) (cit. on pp. 34, 37, 48, 77, 80, 91).

[ML18] A. Mallya and S. Lazebnik. "Packnet: Adding multiple tasks to a single network by iterative pruning". In: *CVPR* (2018) (cit. on p. 59).

[MMV00] E. G. Miller, N. E. Matsakis, and P. A. Viola. "Learning from One Example Through Shared Densities on Transforms". In: *CVPR*. 2000 (cit. on pp. 24, 71).

[NAS18] A. Nichol, J. Achiam, and J. Schulman. "On first-order meta-learning algorithms". In: *CoRR, abs/1803.02999* (2018) (cit. on pp. 6, 23).

[NF16] M. Noroozi and P. Favaro. "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: *ECCV*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. 2016 (cit. on p. 27).

[NHH15] H. Noh, S. Hong, and B. Han. "Learning deconvolution network for semantic segmentation". In: *ICCV*. 2015 (cit. on p. 49).

[Oh +16] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. "Deep metric learning via lifted structured feature embedding". In: *CVPR*. 2016 (cit. on p. 58).

[OLR18] B. N. Oreshkin, A. Lacoste, and P. Rodriguez. "TADAM: Task dependent adaptive metric for improved few-shot learning". In: *arXiv preprint arXiv:1805.10123* (2018) (cit. on pp. 23, 28, 37, 39, 48, 49, 51, 54, 55, 63–66, 77, 80, 91).

[Oqu+14] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks". In: *CVPR*. 2014 (cit. on pp. 20, 22).

[OT06] A. Oliva and A. Torralba. "Building the gist of a scene: the role of global image features in recognition." In: *Progress in brain research* 155 (2006), pp. 23–36 (cit. on p. 3).

[QBL18] H. Qi, M. Brown, and D. G. Lowe. "Low-Shot Learning With Imprinted Weights". In: *CVPR*. 2018 (cit. on pp. 20–22, 25, 31, 42, 48, 49, 65, 66, 115).

[RBV17] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. "Learning multiple visual domains with residual adapters". In: *NIPS*. 2017 (cit. on p. 97).

[Ren+15] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *nips*. 2015 (cit. on pp. 114, 126).

[Ren+18a] M. Ren, R. Liao, E. Fetaya, and R. S. Zemel. "Incremental Few-Shot Learning with Attention Attractor Networks". In: *arXiv preprint arXiv:1810.07218* (2018) (cit. on pp. 77, 80).

[Ren+18b] M. Ren, S. Ravi, E. Triantafillou, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel. "Meta-Learning for Semi-Supervised Few-Shot Classification". In: *ICLR*. 2018 (cit. on pp. 25, 125).

[RJ15] S. Razakarivony and F. Jurie. "Vehicle Detection in Aerial Imagery : A small target detection benchmark". In: *Journal of Visual Communication and Image Representation* (Mar. 2015) (cit. on p. 115).

[RKL16] S.-A. Rebuffi, A. Kolesnikov, and C. H. Lampert. "iCaRL: Incremental Classifier and Representation Learning". In: *arXiv preprint arXiv:1611.07725* (2016) (cit. on pp. 97, 126).

[RL17] S. Ravi and H. Larochelle. "Optimization as a model for few-shot learning". In: *ICLR* (2017) (cit. on pp. 6, 15, 23, 28, 34, 96).

[Rod+20] P. Rodriguez, I. Laradji, A. Drouin, and A. Lacoste. "Embedding Propagation: Smoother Manifold for Few-Shot Classification". In: *arXiv preprint arXiv:2003.04151* (2020) (cit. on pp. 76, 87, 91, 93).

[Rus+14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *arXiv* (2014) (cit. on pp. xxiii, 5, 28).

[Rus+16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016) (cit. on p. 59).

[Rus+18] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. "Meta-learning with latent embedding optimization". In: *arXiv preprint arXiv:1807.05960* (2018) (cit. on p. 39).

[RZL18] P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for activation functions". In: *ICLR* (2018) (cit. on p. 39).

[San+16] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. "Meta-learning with memory-augmented neural networks". In: *ICML*. 2016 (cit. on p. 28).

[Sca+09] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80 (cit. on p. 19).

[Sch+18] E. Schwartz, L. Karlinsky, J. Shtok, S. Harary, M. Marder, A. Kumar, R. Feris, R. Giryes, and A. Bronstein. "Delta-encoder: an effective sample synthesis method for few-shot object recognition". In: *NIPS*. 2018 (cit. on pp. 25, 71).

[Sch87] J. Schmidhuber. "Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook". Diploma Thesis. Technische Universitat Munchen, Germany, 1987 (cit. on p. 15).

[SGS15] R. K. Srivastava, K. Greff, and J. Schmidhuber. "Training Very Deep Networks". In: *NIPS*. 2015 (cit. on pp. 35, 37, 39).

[Shi+18] W. Shi, Y. Gong, C. Ding, Z. M. Tao, and N. Zheng. "Transductive Semi-Supervised Deep Learning using Min-Max Features". In: *ECCV*. 2018 (cit. on p. 76).

[SJT16] M. Sajjadi, M. Javanmardi, and T. Tasdizen. "Mutual exclusivity loss for semi-supervised deep learning". In: *ICIP* (2016) (cit. on p. 76).

[Soh16] K. Sohn. "Improved Deep Metric Learning with Multi-class N-pair Loss Objective". In: *NIPS*. 2016 (cit. on p. 58).

[Son+20] H. Song, M. Kim, D. Park, and J.-G. Lee. "Learning from Noisy Labels with Deep Neural Networks: A Survey". In: *arXiv e-prints* (July 2020) (cit. on p. 125).

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* (2014) (cit. on pp. 27, 65).

[SSZ17] J. Snell, K. Swersky, and R. Zemel. "Prototypical networks for few-shot learning". In: *NIPS*. 2017 (cit. on pp. 16, 19, 20, 23, 25, 26, 28, 31, 34, 35, 46, 51, 55, 60, 62, 65, 87, 88, 91, 104–106, 108, 109).

[Sun+19] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele. "Meta-Transfer Learning for Few-Shot Learning". In: *CVPR*. June 2019 (cit. on pp. 22, 32, 59).

[SZ14] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 35).

[Sze+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *CVPR*. 2015 (cit. on p. 35).

[Tia+20] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola. "Rethinking Few-Shot Image Classification: a Good Embedding Is All You Need?" In: *arXiv preprint arXiv:2003.11539* (2020) (cit. on p. 7).

[TJ18] J. O. du Terrail and F. Jurie. "Faster RER-CNN: application to the detection of vehicles in aerial images". In: *CoRR* abs/1809.07628 (2018) (cit. on p. 114).

[TLF08] E. Tola, V. Lepetit, and P. Fua. "A fast local descriptor for dense matching". In: *CVPR*. June 2008 (cit. on p. 3).

[Tuy10] T. Tuytelaars. "Dense interest points". In: *CVPR*. 2010 (cit. on p. 3).

[TV17] A. Tarvainen and H. Valpola. "Mean Teachers Are Better Role Models: Weight-Averaged Consistency Targets Improve Semi-Supervised Deep Learning Results". In: *NIPS*. 2017 (cit. on p. 76).

[VC17] P. Vernaza and M. Chandraker. "Learning Random-Walk Label Propagation for Weakly-Supervised Semantic Segmentation". In: *CVPR*. 2017 (cit. on p. 83).

[Ver+19] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio. "Manifold Mixup: Better Representations by Interpolating Hidden States". In: *ICML*. 2019 (cit. on p. 27).

[Vin+16] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. "Matching networks for one shot learning". In: *NIPS*. 2016 (cit. on pp. 18, 19, 25, 28, 31, 35, 46, 77, 80, 87, 91).

[Wah+11] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. *The Caltech-UCSD Birds-200-2011 Dataset*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011 (cit. on p. 28).

[Wan+17a] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. "Normface: l 2 hypersphere embedding for face verification". In: *ACM Multimedia*. 2017 (cit. on p. 49).

[Wan+17b] J. X. Wang, Z. Kurth-Nelson, H. Soyer, J. Z. Leibo, D. Tirumala, R. Munos, C. Blundell, D. Kumaran, and M. M. Botvinick. "Learning to reinforcement learn". In: *ArXiv* abs/1611.05763 (2017) (cit. on p. 15).

[Wan+17c] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin. "Deep metric learning with angular loss". In: *ICCV*. 2017 (cit. on p. 58).

[Wan+18a] W. Wan, Y. Zhong, T. Li, and J. Chen. "Rethinking feature distribution for loss functions in image classification". In: *CVPR*. 2018 (cit. on p. 6).

[Wan+18b] Y.-X. Wang, R. Girshick, M. Hebert, and B. Hariharan. "Low-Shot Learning from Imaginary Data". In: *CVPR*. 2018 (cit. on p. 15).

[Wan+19] Y. Wang, W.-L. Chao, K. Q. Weinberger, and L. van der Maaten. "SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning". In: *arXiv preprint arXiv:1911.04623* (2019) (cit. on pp. 7, 21, 31).

[Wan+20a] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. "Generalizing from a Few Examples: A Survey on Few-Shot Learning". In: *ACM Comput. Surv.* 53.3 (June 2020) (cit. on p. 13).

[Wan+20b] Y. Wang, C. Xu, C. Liu, L. Zhang, and Y. Fu. "Instance Credibility Inference for Few-Shot Learning". In: *arXiv preprint arXiv:2003.11853* (2020) (cit. on p. 91).

[Wel+10] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. *Caltech-UCSD Birds 200*. Tech. rep. CNS-TR-2010-001. California Institute of Technology, 2010 (cit. on p. 96).

[WH19] D. Wertheimer and B. Hariharan. "Few-Shot Learning With Localization in Realistic Settings". In: *CVPR*. 2019 (cit. on pp. 61, 77, 80, 81).

[Wil+04] J. Willamowski, D. Arregui, G. Csurka, C. R. Dance, and L. Fan. "Categorizing Nine Visual Classes Using Local Appearance Descriptors". In: *In ICPR Workshop on Learning for Adaptable Visual Systems*. 2004 (cit. on p. 3).

[Wu+19] Z. Wu, Y. Li, L. Guo, and K. Jia. "PARN: Position-Aware Relation Networks for Few-Shot Learning". In: *ICCV*. 2019 (cit. on pp. 77, 80).

[XHS18] Z. Xu, H. van Hasselt, and D. Silver. "Meta-Gradient Reinforcement Learning". In: *NIPS*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 2402–2413 (cit. on p. 15).

[Xv+19] H. Xv, X. Sun, J. Dong, S. Zhang, and Q. Li. "Multi-level Similarity Learning for Low-Shot Recognition". In: *arXiv preprint arXiv:1912.06418* (2019) (cit. on pp. 77, 80).

[Yan+18] F. S. Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. "Learning to compare: Relation network for few-shot learning". In: *CVPR*. 2018 (cit. on pp. 17, 35).

[Yan+20] L. Yang, L. Li, Z. Zhang, E. Zhou, Y. Liu, et al. "DPGN: Distribution Propagation Graph Network for Few-shot Learning". In: *arXiv preprint arXiv:2003.14247* (2020) (cit. on p. 92).

[Ye+18] H. Ye, H. Hu, D. Zhan, and F. Sha. "Learning Embedding Adaptation for Few-Shot Learning". In: *CoRR* abs/1812.03664 (2018) (cit. on p. 29).

[Yoo+18] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. "Lifelong Learning with Dynamically Expandable Networks". In: (2018) (cit. on p. 97).

[Yos+14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *NIPS*. 2014 (cit. on pp. 59, 97).

[ZG02] X. Zhu and Z. Ghahramani. *Learning From Labeled and Unlabeled Data with Label Propagation*. Tech. rep. Carnegie Mellon University, 2002 (cit. on pp. 78, 83).

[Zha+20] C. Zhang, Y. Cai, G. Lin, and C. Shen. "DeepEMD: Few-Shot Image Classification With Differentiable Earth Mover's Distance and Structured Classifiers". In: *CVPR*. 2020 (cit. on pp. 77, 80, 92).

[Zho+03a] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. "Learning with local and global consistency". In: *NIPS*. 2003 (cit. on pp. 78, 83–85, 87).

[Zho+03b] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. "Ranking on Data Manifolds". In: *NIPS*. 2003 (cit. on p. 83).

[Zho+16] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. "Learning Deep Features for Discriminative Localization". In: *CVPR*. 2016 (cit. on pp. 49, 50, 82).

[Zho+17] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. "Places: A 10 million Image Database for Scene Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017) (cit. on p. 101).

[ZK16] S. Zagoruyko and N. Komodakis. "Wide Residual Networks". In: *BMVC*. 2016 (cit. on pp. 35, 39).

[ZZK19] H. Zhang, J. Zhang, and P. Koniusz. "Few-Shot Learning via Saliency-Guided Hallucination of Samples". In: *CVPR*. 2019 (cit. on pp. 77, 80, 81).

List of published contributions

[LAP20a] Y. Lifchitz, Y. Avrithis, and S. Picard. “Few-Shot Few-Shot Learning and the role of Spatial Attention”. In: *ICPR* (2020) (cit. on pp. 58, 97).

[LAP20b] Y. Lifchitz, Y. Avrithis, and S. Picard. “Local Propagation for Few-Shot Learning”. In: *ICPR* (2020) (cit. on p. 77).

[Lif+19] Y. Lifchitz, Y. Avrithis, S. Picard, and A. Bursuc. “Dense Classification and Implanting for Few-Shot Learning”. In: *CVPR* (2019) (cit. on pp. 34, 58, 91).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Exploiter au mieux les données disponibles : représentation et adaptation pour la classification few-shot d'images

Mots-clés : Few-shot, apprentissage automatique, profond, representation, adaptation, vision

Résumé : Les réseaux de neurones profonds peuvent être entraînés pour produire des modèles de classification d'images très précis, à condition d'avoir accès à un grand nombre de données d'apprentissage. Dans le cas du few-shot learning, les données sont limitées à quelques images ce qui ne permet pas l'apprentissage complet. Dans un premier temps, une fonction de représentation indépendante de la tâche est apprise en résolvant une tâche distincte comme la classification des classes de base. Ensuite, la représentation est combinée avec des exemples des nouvelles classes pour résoudre la tâche few-shot. Pour les deux étapes, nous introduisons des solutions exploitant au mieux les données disponibles. Pour l'apprentissage de représentation, nous proposons la classification dense, qui étudie pour la première fois les activations locales pour le few-shot learning. De plus, nous proposons deux solutions pour adapter la fonction de représentation à la tâche few-shot. L'apprentissage est limité à quelques paramètres dans le cas de l'implantation, ou à quelques itérations. Nous étudions également des problèmes de few-shot learning pour lesquels l'accès à l'information est modifié. Dans le cas du few-shot transductif, plusieurs images doivent être classifiées en même temps. Nous proposons la propagation locale, utilisant les similarités entre représentations locales pour propager l'information de classe. Nous proposons également un nouveau problème, le few-shot few-shot learning, où peu ou aucunes données du domaine n'est accessible. On peut utiliser un réseau pré-entraîné en l'adaptant si possible avec des données du modèle. Pour le few-shot learning, il est important de se focaliser sur les régions pertinentes des images. Nous proposons deux solutions simples d'attention. Enfin, nous appliquons notre savoir dans le cas spécifique de la classification d'images aériennes.

Title: Making the most of available data: representation and adaptation for few-shot image classification

Keywords: Few-shot, machine learning, deep learning, representation, adaptation, vision

Abstract: Deep neural networks can be trained to create highly accurate image classification models, provided we have access to large datasets. In few-shot learning, data is limited to few images, so training from scratch is not feasible. First, a task-independent representation function is learned on abundant data by solving a distinct task such as multi-class classification on a set of base classes. Then, the learned representation is combined with new data of novel classes to solve the few-shot task. In both stages, we introduce solutions that aim at leveraging available data as much as possible. In particular, for representation learning, we propose dense classification training, which for the first time studies local activations in the domain of few-shot learning. We also propose two solutions to adapt the representation function to the few-shot task. Learning is limited to a few parameters in implanting or to few gradient updates. Additionally, we study alternative few-shot learning settings, in which access to data is modified. In transductive learning, multiple images need to be classified at the same time. In this context, we propose local propagation, a method that uses similarities between local representations of images to propagate class information. We also introduce few-shot few-shot learning, a new setting, where only few or no in-domain data is accessible for representation learning. In this context, we take advantage of a classifier, pre-trained on a large-scale dataset of a different domain, which can still be adapted to the domain if data is available. In few-shot learning, because data is so scarce, we show that selecting relevant regions with an attention mechanism is important. We propose two simple solutions that successfully fulfill this role. Finally, we apply our knowledge of few-shot learning on the specific problem of classifying aerial images.