Graphs as Tools to Improve Deep Learning Methods

Carlos Lassance, Myriam Bontonou, Mounia Hamidouche, Bastien Pasdeloup,
Lucas Drumetz, Vincent Gripon

Graphs as Tools to Improve Deep Learning Methods

Carlos Lassance

Myriam Bontonou

Mounia Hamidouche

Bastien Pasdeloup

Lucas Drumetz

Vincent Gripon

Chapter 1

Graphs as Tools to Improve Deep Learning Methods

1.1   Introduction

In recent years, deep neural networks (DNNs) have known an important rise in popularity. A milestone was reached in 2012, when for the first time, a DNN called AlexNet [29] won the CVPR LSRV challenge [48] by a large margin. Since then, outstanding performance on many important problems have been achieved with DNNs, particularly in the fields of computer vision [15] [29], natural language processing [58] and gaming [51]. Other domains such as text translation [24], speech to text [61], person identification and verification [17], help in screening and diagnosis in medicine [7], object detection [63], user behavior study [38] or even art restoration [12] have also gained momentum in the past few years. This performance of DNN systems in a large number of application domains is largely due to the accessibility of large training datasets and to large computational processing resources as with GPUs [13].

DNN models are thus state-of-the-art in many machine learning challenges, even if they have

limitations. For example, DNNs require a lot of training data, which might not be available in some practical applications. In addition, even though trained models seemingly perform well, if small perturbations are added to their inputs, they are prone to misclassification errors [53]. This susceptibility can be critical in some applications such as self-driving vehicles. Furthermore, DNN models are often viewed as "black-boxes" and as such their decisions are often criticized for their lack of *interpretability* [56]. Indeed, there are a lot of domains that would benefit from explainable models, among which product-recommendation, targeted advertising, medical diagnosis...

There is a clear explanation to the above-mentioned shortcomings. Indeed, DNNs were introduced as a mean to replace "hand-crafted" features with learned ones. This is performed thanks to the use of automatic differentiation to train a model end-to-end, meaning that only the output is explicitly constrained during the learning process. And yet DNNs are compositional models obtained by assembling elementary functions called *layers*. As such, they produce multiple intermediate representations when processing an input element. The intermediate representations are rarely explicitly constrained during training with the intention of removing any arbitrary human intervention. However, giving more latitude to the training process brings the disadvantage of harder interpretability, unwanted behaviors and more data requirements.

In recent years, authors have shown that processing multiple inputs at once can help in designing more efficient models. This is the key ingredient behind the popular *batch-normalization* [20] layer. Batch-normalization introduced the promising idea of constraining the intermediate representations of a batch of inputs relatively to each other. As such, there is no explicit constraint on a single representation, that is still freely optimized during the training phase. But multiple representations are constrained to exhibit certain behaviors, such as competition in the case of batch-normalization. More generally, exploiting the relations between different samples of the training set is a direction of research to address the above-mentioned problems.

In this chapter, we review recent works that aim at using graphs as tools to improve deep learning methods. These graphs are defined considering a specific layer in a deep learning architecture. Their vertices represent distinct samples, and their edges depend on the similarity of the corresponding intermediate representations. The same principle can be deployed to define graphs at the input, or at the output of deep learning architectures. These graphs can then be leveraged using various methodologies, many of which built on top of graph signal processing (GSP) [50].

The outline of this chapter is as follows. We first introduce the needed concepts from DNN and GSP frameworks in Section 1.2. We then review the recent contributions aiming at using graphs to improve deep learning methods in Section 1.3. This Section is composed of four main parts: tools for visualizing intermediate layers in a DNN, denoising data representations, optimizing graph objective functions and regularizing the learning process. Section 1.4 provides some concluding remarks.

1.2 Background

1.2.1 Deep Neural Networks for Computer Vision

In this section, we introduce the fundamental aspects about DNNs that will be of use in the remaining of the document.

A DNN is a mathematical function f that associates an input $\mathbf{x}$, typically in a multidimensional array, with a corresponding output $\hat{\mathbf{y}} = f(\mathbf{x})$. It is obtained by assembling elementary functions that are called *layers* in the literature. In the simplest case, this assembly consists in a simple composition of the layer functions, so that the DNN can be mathematically described as

$$f = f^{\ell_{\max}} \circ f^{\ell_{\max}-1} \circ \cdots \circ f^1, \tag{1.1}$$

where each f^i is a layer function. In more complex cases, it is often the case that a DNN function can be decomposed as $f = g \circ h$, where g and h are possibly complex assemblies themselves.

When processing an input element $\mathbf{x}$, it is therefore possible to decompose two parts $\mathbf{z} = h(\mathbf{x})$ and $\hat{\mathbf{y}} = g(\mathbf{z})$. In such a case, we call $\mathbf{z}$ an *intermediate representation* associated with $\mathbf{x}$. In practice there are multiple intermediate representations associated with an input $\mathbf{x}$ for multiple layers in the considered DNN.

Most layers contain trainable *parameters*. Their values are initialized at random and updated during a training phase. This phase usually consists in minimizing a loss function, which is a function of the discrepancy between the actual output $\hat{\mathbf{y}} = f(\mathbf{x})$ of the DNN and the expected one $\mathbf{y}$. This loss is minimized on a *training set*. In the literature, many architecture designs f can be found, some of which are particularly popular. One example is the residual network (ResNet) [15], that typically achieves good performance with a limited design complexity.

As various designs exist, it is common to rely on cross validation to choose the one that yields the best performance. In that case, the training set is split into two parts prior to training. The first part is used as a new training set for the DNN. The second part, called the *validation set* is used to pre-evaluate the ability of the trained model to generalize to new data. More precisely, at the end of the training, we obtain the final architecture with the corresponding parameters. Then, one evaluates the accuracy of the DNN on the validation set. The accuracy is the number of correctly predicted data points out of all the validation set. The chosen design is the one that generalizes better. When the error on the training set is small, whereas the error on previously unseen inputs is high, the DNN is *overfitting*. To assert that the chosen DNN does not overfit and can be used to process previously unseen inputs, it is evaluated on a third dataset, called *test set*.

Limitations of DNNs

Considering a large amount of labeled data, and extensive computational power, DNNs are in many cases the golden standard of machine learning problems. However, major limitations refrain the adoption of these methods in specific domains. This chapter is built around four limitations that are all tackled using graph-based representations.

The first limitation is the **lack of interpretability** of the decisions taken by DNNs. This is problematic for applications that aim at assisting human experts (e.g. assisted medical diagnosis). Since it is difficult to understand how the DNN processes information, its ability to take a good decision on new inputs is assessed via the test set. However, a DNN performing well on a test set is not guaranteed to perform well on slightly different or unexpected inputs. Various methods have been proposed to visualize the features that matter in the decision process, both in the inputs and in the learned intermediate representations [1, 47, 54]. To directly measure the ability of generalization, without necessarily needing a validation set, some metrics have also been proposed [5, 11, 22, 23]. In Subsection 1.3.1, graph-based measures are introduced to assess the ability of generalization of DNNs.

The second limitation is the difficulty to **train models with little data**. In the literature, learning on very few labeled examples is called *few-shot learning*. More generally, we a few-label is a task that is described as N-way K-shot U-unlabeled, where N is the number of classes to distinguish within data, K the number of labeled images available per class during training, and U unlabeled images per class may be available for training. In the case of a few-shot task U is always equal to 0.

Many popular solutions for few-shot problems rely on *knowledge transfer* [55, 62]. The core idea is to exploit a DNN $f' = g' \circ h'$ trained on a distinct, yet similar and larger dataset, for the newly considered problem. More precisely, instead of training a new DNN f directly on raw inputs $\mathbf{x}$, a function g, taking $h'(\mathbf{x})$ as its input, is trained. In such a case, h' is called a *feature extractor* and g a *classifier*, $f = g \circ h'$ is therefore the new DNN function. The presented approach is preferred when the small dataset is very limited [3, 41]. If more data is available, it is also common to *fine tune h'* as well by training f for a few epochs [27]. An epoch refers to a pass of the gradient descent algorithm through the entire training set. Subsection 1.3.2 introduces ways of using graphs to help denoising the representations learned by the feature extractors, in order to better handle the underlying problem.

The third limitation lies on the fact that **the output of the DNN is most of the time independent of the distribution of the input and of the initialization of the network parameters**, which can slow down and complicate the training process. To understand this limitation, consider the cross-entropy loss function, the most popular loss function for classification problems in computer vision. In this case, the dimension of the output vectors has to be equal to the number of classes, preventing an easy adaptation to the introduction of new classes. In scenarios where the number of classes is large, this also causes the last layer of the network to contain a lot of parameters. Moreover, the cross-entropy loss considers that it is possible to completely separate the classes but does not take into account a possible hierarchy or similarity between classes. To overcome these drawbacks, one alternative to this is to optimize the embeddings directly, without forcing arbitrary separation such as the case of the siamese [26], triplet [17] and smoothness losses [4]. This limitation is addressed in Subsection 1.3.3.

The last considered limitation is the **lack of robustness** that occurs when DNN architectures are susceptible to deviations of their inputs and thus perform poorly [10, 54]. The lack of robustness may cause errors in decisions in various practical settings, in which data is prone to noise or manipulations. When discussing robustness of DNNs, it is common to separate two sources of deviations. The first

source of deviation is the one that is not performed intentionally to fool the DNN [16][1]. Examples of such deviations range from defects in the data capture (e.g. Gaussian noise) to different luminosity conditions (e.g. brightness and contrast). The second deviation is the one that is created explicitly to fool the network, also called adversarial attacks. In this case, a metric is used to measure the distance between the original image and the adversarial one [10]. A successful adversary is the one that fools the DNN, without trespassing a perceptibility threshold on the metric (e.g. the adversary does not change any individual pixel color of more than 8/255 [39]). This limitation is addressed in Subsection 1.3.4.

1.2.2 Graph Signal Processing

Graph Signal Processing (GSP) offers a convenient framework for studying data while taking into account the complex domain on which they are defined. This section introduces the main tools that are of use in GSP. In the remainder of this chapter, we will use them to express desirable constraints on DNNs.

Graphs and Graph Signals

Consider a dataset comprising n samples, and suppose that there exist specific relations between them. For example, samples can be individuals in a social network and relations are given by their friendship connections. A concise and convenient way to model samples and their relations is to use the formalism of graphs.

A *graph* $\mathcal{G}$ is a tuple of sets $\langle \mathbb{V}, \mathbb{E} \rangle$, where $\mathbb{V}$ is composed of *vertices* indexed from 1 to n: $\mathbb{V} = \{v_1, \ldots, v_n\}$, and $\mathbb{E}$ is composed of pairs of vertices of the form (v_i, v_j) called *edges*. These edges are unordered, such that (v_i, v_j) refers to the same pair as (v_j, v_i).

It is common to represent the set $\mathbb{E}$ using an edge-indicator symmetric *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{|\mathbb{V}| \times |\mathbb{V}|}$. When relations are weighted, this matrix can take values other than 0s and 1s.

In a graph, some vertices might have more importance than others. A common way to measure importance of vertices is to define their *degree* (or *strength* when graphs are weighted). The degree of a vertex v is simply obtained by summing the weights of all edges v is part of. These quantities can be assembled in the diagonal *degree matrix* $\boldsymbol{D}$:

$$D_{i,j} = \begin{cases} \displaystyle\sum_{j' \in \mathbb{V}} A_{i,j'} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \qquad . \tag{1.2}$$

Now let us suppose that each of the n samples is associated with a vector of dimension d. This can be represented as a $n \times d$ matrix. Columns of this matrix are called graph signals. For example, the age of each individual in a social network can be viewed as a graph signal.

As a natural representation of complex data structures, graphs and graph signals are ubiquitous, in particular in the field of machine learning. In the following paragraphs we introduce operations on graphs that are going to be useful in the rest of this chapter.

---

[1]such deviations are called *corruptions* in this chapter.

Graph Fourier Transform (GFT)

In the domain of signal processing, *frequencies* are one of the most important concepts to analyze a signal and a starting point to introduce many useful tools such as filtering. The frequency of a signal can be simplified as its rate of change, or in other words how it varies from one sample to another. Generally speaking, any signal can be decomposed into a (possibly continuous) sum of sines/cosines of various frequencies, by performing the celebrated Fourier transform. This decomposition can be understood as expressing the signal in the frequency domain, providing a dual representation of the original signal in the time domain. In other words, the Fourier transform of a signal is a simple change of basis, that transports a signal to a convenient domain for many signal processing operations.

Graph signal processing arose as a graph-centric generalization of the classical Fourier analysis [50]. In this framework, a ring or a path-shaped graph is used to model the time domain, in which each vertex models an instant in time where the intensity of the signal is sampled. One therefore manipulates two separate objects: a graph $\mathcal{G}$ modeling time – the support of information – and a graph signal s – the sampled information –. In a path graph, each vertex is connected with at most one subsequent vertex (in this case the next sample in the time domain) and at most one previous vertex (in this case the previous sample in the time domain). This means that in a path graph there exist two vertices with only one edge, called "first" and "last". A regular ring graph is then defined as a path graph where the "first" vertex is connected to the "last" (one-dimensional lattice), forming a cycle with periodic boundary conditions, that is very much a natural subject for Fourier analysis.

The core observation of GSP is that Fourier transform is closely tied to the domain over which signals evolve, and not to the signals themselves. More precisely, consider a graph whose adjacency matrix only contains nonnegative values. Its combinatorial Laplacian L is defined as:

$$L = D - \mathbf{A} \ . \tag{1.3}$$

As the Laplacian matrix is both real and symmetric it can be eigendecomposed into:

$$L = F\Lambda F^\top \ , \tag{1.4}$$

where the columns of F are the eigenvectors of L and the diagonal coefficients of Λ are the eigenvalues in increasing order of magnitude.

If we consider the regular ring graph mentioned above that has a periodic property to model time as a support of information, one can show that the eigenvectors in F are described as sine/cosine functions (Fourier series), with increasing frequency as associated eigenvalues increase. The Fourier transform of a graph signal s is thus simply a change of basis defined by F, and can be written as:

$$\hat{s} = F^\top s \ . \tag{1.5}$$

This projection of s in the eigenbasis F of L is what we call the *Graph Fourier Transform (GFT)* of s. Note that the inverse GFT can be similarly defined as:

$$s = F\hat{s} \ . \tag{1.6}$$

GFT obviously depends on the graph that is used to model the signals under study. As a consequence, the choice of the underlying graph is of paramount importance. A path (or ring) graph provides a comprehensive domain to study (periodic) time signals, but it is often not clear which graph should be used for more complex signals. However, it still holds that eigenvectors $\boldsymbol{F}$ of the Laplacian matrix of a graph characterize the variations over that graph, with increased local variations as the associated eigenvalues increase. The interested reader can refer to [50] and [44] for illustrative examples.

Exploiting this property, authors have defined many tools generalizing signal processing operations on temporal signals, such as convolution, filtering, translation or modulation [50]. We present in the next sections some of the uses of the GFT relevant to the works introduced in this chapter.

Smoothness of graph signals A measure that will be at the core of many of the tools we introduce in the next section is the *graph smoothness*. Let $\mathcal{G}$ be a graph of adjacency matrix $\mathbf{A}$, and let $\boldsymbol{L}$ be its combinatorial Laplacian, with eigenvalues $\boldsymbol{\Lambda}$. The smoothness $\sigma(\boldsymbol{s})$ of a graph signal $\boldsymbol{s}$ supported on $\mathcal{G}$ is defined as:

$$\sigma(\boldsymbol{s}) = \boldsymbol{s}^\top \boldsymbol{L} \boldsymbol{s} = \sum_{i=1}^{|\mathbb{V}|} \Lambda_{i,i} \hat{\boldsymbol{s}}_i^2 = \sum_{i=1}^{|\mathbb{V}|} \sum_{j=1}^{|\mathbb{V}|} A_{i,j} (\boldsymbol{s}_i - \boldsymbol{s}_j)^2 \,. \tag{1.7}$$

Lower values of the smoothness metric are said to be very smooth with respect to the graph, and higher values of the smoothness metric are said to be very rough (or unsmooth) with respect to the graph. Equation (1.7) shows that a smooth signal is a signal aligned with the first eigenvectors of the Laplacian. In addition, we observe that the smoothness is strongly related to the rate of change of the signal values from one vertex to its neighbors.

Considering the smoothness as the rate of change in a signal $\boldsymbol{s}$ across a graph, it is a very useful abstraction, even more when $\boldsymbol{s}$ is binary. In this case, the smoothness can be simplified as the sum of the weights between neighboring nodes with different values in $\boldsymbol{s}$.

Consider an illustrative case where vertices represent samples in a classification dataset. An interesting signal is the label indicator vector of a chosen class, which is a binary vector that contains a 1 at coordinate i if i is of the considered class and 0 otherwise. In such a case, the smoothness boils down to the sum of the weights of edges connecting pairs where one element is of the considered class and the other is not. Therefore, the only graphs that would nullify smoothness of this signal are those that contain no edge between samples of the class and samples of a distinct class.

Graph filters In classical discrete signal processing, filters are tools meant to act on signals by removing specific ranges of frequencies. In the following, we develop the equivalent concept of graph filters for graph signals in GSP.

Here, we introduce three possible representations of these filters. For each representation, we provide its advantages/drawbacks and detail some of their applications. We refer the reader to [14, 57] for a more in depth discussion on graph filters.

1) Design of filters in the spectral domain The simplest and most general way to define a filter is to describe its response for each frequency. In the case of graph signals, frequencies are defined by the diagonal values of the eigenvalue matrix $\boldsymbol{\Lambda}$. For notation simplicity we consider the frequency

vector $\boldsymbol{\lambda}$ where $\lambda_i = \Lambda_{i,i}$, sorted in increasing order of magnitude.

We define a filter on a graph $\mathcal{G}$ using a diagonal matrix $\boldsymbol{H}_\mathcal{G} \in \mathbb{R}^{|\mathbb{V}| \times |\mathbb{V}|}$, where we call each element $H_{i,i}$ the response of the filter to frequency λ_i. The filter can then be convoluted with a graph signal $\boldsymbol{s}$ as in classical signal processing, by performing a multiplication in the spectral domain [50]:

$$\tilde{\boldsymbol{s}} = \boldsymbol{F}\boldsymbol{H}_\mathcal{G}\boldsymbol{F}^\top\boldsymbol{s} \ . \tag{1.8}$$

Defining filters directly in the spectral domain generates filters that are graph-specific. Indeed, as the frequencies λ_i are discrete, the same filter $\boldsymbol{H}$ will be applied to very different frequencies for two distinct graphs $\mathcal{G}$ and $\mathcal{G}'$. Therefore this type of filter tends to be mostly used to remove the lowest or highest frequencies of the graph, without considering their "true" value. Another drawback is that it is unlikely that this type of filter may be represented with a low order polynomial filter which impacts the complexity (i.e. the possibility of scaling to larger graphs) of applying the filter.

2) Design of filters using their spectral response

Another straightforward way to define a filter is by its spectral response, i.e. a function of the frequency. In this way the filter becomes less graph-dependent and more general. One such design is the Simoncelli filter, depicted in Figure 1.1, where $\tau \in [0,1]$ is a user-defined threshold and λ_i the i-th Laplacian eigenvalue.

Defining a filter by its spectral response allows for more universal filters (i.e. filters that do not heavily depend on the graph support) and also to easily represent the filter by a low order polynomial function. In this work, we use the PyGSP [9] toolbox to implement this type of graph filters, which uses the Chebyshev polynomial approximation in order to apply the filters.

$$h(\lambda_i) = \begin{cases} 1 & \text{if } \frac{\lambda_i}{\lambda_{|\mathbb{V}|}} \leq \frac{\tau}{2}, \\[2ex] \cos\left(\frac{\pi}{2}\frac{\log\left(\frac{\lambda_i}{\tau\lambda_{|\mathbb{V}|}}\right)}{\log(2)}\right) & \text{if } \frac{\tau}{2} < \frac{\lambda_i}{\lambda_{|\mathbb{V}|}} \leq \tau, \\[2ex] 0 & \text{if } \frac{\lambda_i}{\lambda_{|\mathbb{V}|}} > \tau. \end{cases}$$
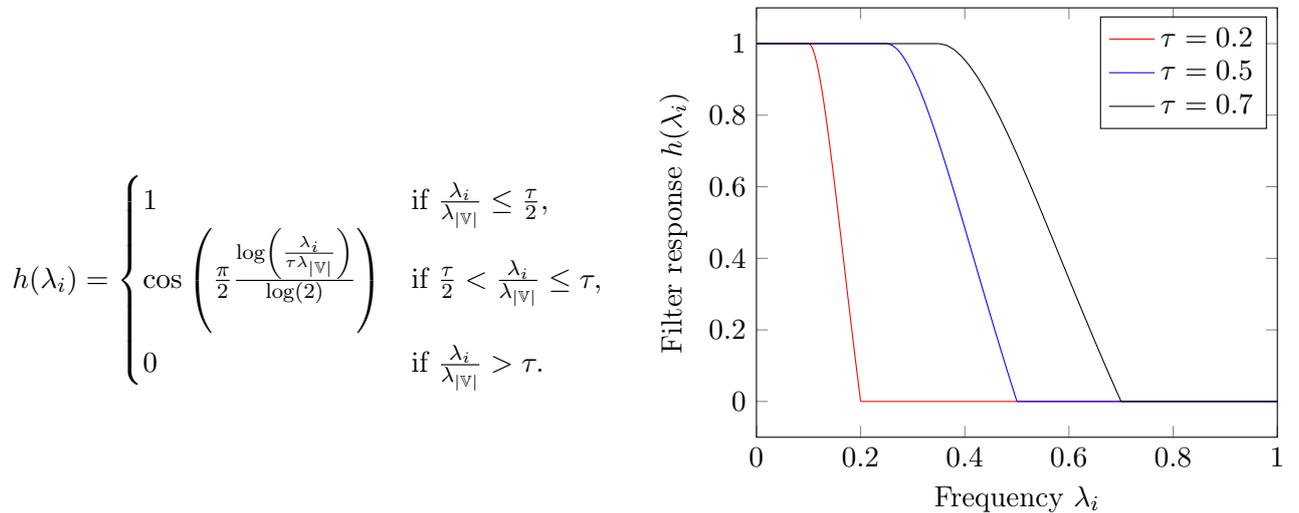
Figure 1.1: Equation (left) and depiction (right) of the Simoncelli filter spectral response for various values of τ.

3) Design of filters using diffusion operators

It is also possible to define a graph filter using a diffusion operator $\boldsymbol{S} \in \mathbb{R}^{|\mathbb{V}| \times |\mathbb{V}|}$, sometimes also called *graph shift operator*. A diffusion operator on a graph $\mathcal{G}$ is a matrix $\boldsymbol{S}$ describing the same edge set as the adjacency matrix $\boldsymbol{A}$ of $\mathcal{G}$, i.e. such that $S_{i,j} = A_{i,j}$ where $A_{i,j} = 0$. It allows to define the

notion of information flow over a graph, as it represents one elementary discrete step on how the information propagates (shifts) from one node to its neighbors. In classical signal processing – where the graph structure can be modeled as a directed cycle graph – the information flow is unidirectional, and goes from each node to its only neighbor. In a more complicated graph structure, the information flow is neither restricted to unidirectional structure nor to a limited number of physical neighbors but depends on the graph adjacency matrix. When information flows to a vertex from all the vertices it is neighbor of, incoming signal is combined through a weighted sum as follows [49]:

$$\tilde{s} = Ss \ . \tag{1.9}$$

Note that by being applied with just a simple matrix multiplication, this type of filter is easily integrated in deep learning scenarios, where matrix multiplication is prevalent. Indeed, most of the recent developments in graph convolutional layers use this design.

Example of application of GSP

Here is an illustrative example showing the usefulness of the notion of graph filters and smoothness. Figure 1.2 depicts a graph signal on a path graph, a noisy version of it, and a low-pass filtered version of the noisy graph signal. Note that both the original graph signal and the filtered version are smoother than the noisy version of the signal. Thus, the smoothness of a graph signal can also be used to detect the presence of noise in the signal.


(a) $\sigma(s) = 0.132$ (b) $\sigma(\check{s}) = 1270.902$ (c) $\sigma(\tilde{s}) = 0.144$
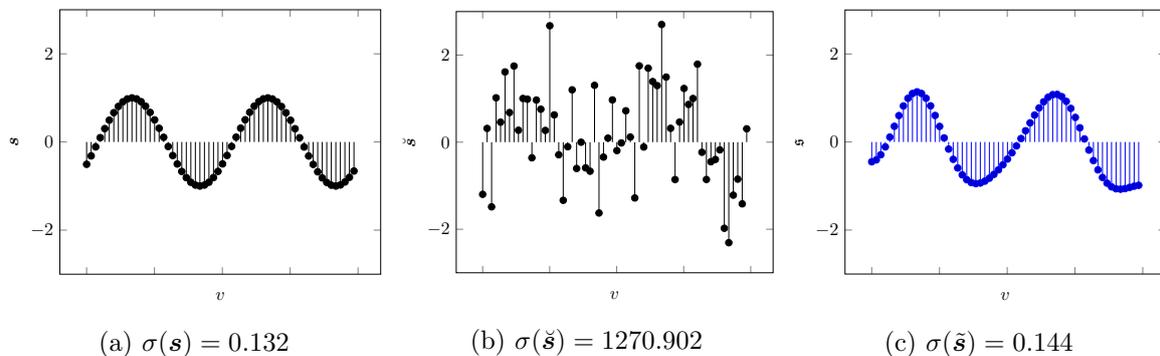
Figure 1.2: Depiction of a sampled signal s, its noisy version $\check{s}$ and its filtered version $\tilde{s}$, with their respective smoothness values.

In addition, here is an example of a picture, seen as a signal of dimension 3 (RGB) over a grid graph. Using a low-pass graph filter, it is possible to exploit the structure of the pixels to remove noise from the image as shown in Figure 1.3. By removing the high frequencies of the graph signal, the resultant image is more in line with the original image. Note also that the smoothness value of the recovered graph signal is the same as the original image, while the images are very different. Indeed as the smoothness of a graph signal is a global measure and not a localized one it is easy to see that multiple image configurations are possible for the same value of smoothness.

(a) $\sigma \equiv 1211 \times 10^4$ (b) $\sigma \equiv 2043 \times 10^4$ (c) $\sigma \equiv 1217 \times 10^4$
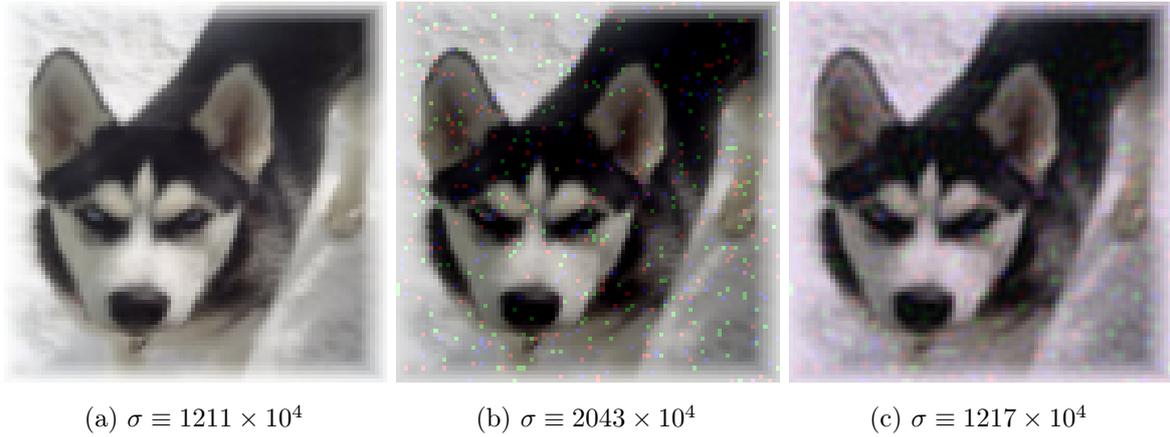
Figure 1.3: Depiction of a dog (left), a noisy realization of the image (center) and the graph filtered image (right).

1.3   Graphs as Tools to Improve Deep Learning Methods

Now that we have introduced DNNs, some of their limitations and presented the framework of GSP, we delve in more details into how to improve deep learning methods using graphs.

1.3.1   Graphs to Interpret Intermediate Representations

Using smoothness to measure the state of DNNs

Throughout the training process of a DNN, an input vector is transformed into multiple intermediate representations of various dimensions and shapes. Understanding how these intermediate representations evolve during the training process is a major open question.

In particular, it is difficult to know whether a trained architecture is able to solve the problem it was trained for or not. To address this limitation, cross validation is often performed to measure the state of a DNN: underfitted, overfitted or correctly optimized. However, in many practical cases, it is problematic to rely on cross validation. Indeed, the training set is reduced in size, requiring to tune parameters with a limited number of examples. Also, the model generalization abilities are assessed using a fraction of the training set, giving a possibly biased judgement on their real performance on unseen data.

A more principled approach was proposed in [11] where the authors propose to use GSP to monitor the training of DNNs. Their method is based on the notion of smoothness of graph signals. Namely, a graph $\mathcal{G}$ is constructed at different layers of the DNN. Its vertices are the intermediate representations of the training samples and its edges are the similarities between these samples. The graph signal $\boldsymbol{s}^c \in \mathbb{R}^{|\mathbb{V}|}$ indicates whether the vertices belong to the class c. More precisely, given a vertex i, $\boldsymbol{s}_i^c = 1$ if the class of the i-th sample is c and 0 otherwise.

Let M be the number of examples in each class c, C be the number of classes and $\boldsymbol{L}$ the Laplacian of $\mathcal{G}$. For a given graph signal $\boldsymbol{s}^c$, the label smoothness is then defined as:

$$Smooth = \frac{\sum_{c=1}^{C} \boldsymbol{s}_c^\top \boldsymbol{L} \boldsymbol{s}_c}{M^2 C(C-1)}.$$

The term $M^2(C-1)$ is a normalization factor. The label smoothness can be rewritten as the sum of the similarities between samples of distinct classes. Thus, the label smoothness is 0 if and only if the similarity between samples of distinct classes is 0.

The authors of [11] were able to show that the label smoothness can clearly differentiate different states of trained DNNs. More precisely, they observed that when the considered architecture is correctly optimized, the values of label smoothness on the last layers of the trained architecture are similar, whereas in the other conditions, they are separated by important gaps.

They also observed that during training, the label smoothness continues to evolve when the training accuracy stabilizes around 100%. Indeed, this motivates the idea that even if the training accuracy is maximal, the intermediate representations are still changing. Therefore, the label smoothness could be used as a measure to stop or continue the DNN training when the training accuracy has reached its maximal value.

Detecting the influential instances

To better interpret a DNN, one can look at many different variables, e.g. the training examples that influence the most the predictions or the most influential pixels inside an image. In this subsection, a method called *model analysis and reasoning using graph-based interpretability* (MARGIN) [1] is presented. In this method, the studied influential variables are called *influential instances*. The method is divided into four steps:

1. *Influential instances*: First, the domain on which the analysis is performed is described. It can be the set of training samples, where each instance is an image, or even a single image, where a group of neighboring pixels is an instance.

2. *Graph construction*: A graph $\mathcal{G}$ modeling the relationships between the instances is constructed. The vertices are the instances and the edges are the similarities between the instances. The similarities can be computed from the raw instances or, when applicable, from intermediate representations of the instances within the DNN.

3. *Graph signal*: The graph signal represents the source of variations on which the influence of each instance is studied. For example, considering that an instance is a training example, the graph signal could be a measure indicating how much the neighbors of the example in $\mathcal{G}$ disagree with its label.

4. *Sample-wise influence estimation*: MARGIN aims at finding the vertices of the graph $\mathcal{G}$ that characterize the most the variations of the signal with respect to the structure of the graph $\mathcal{G}$. Using the formalism of GSP, a high-pass filter is applied over the graph signal. The influence of each vertex is proportional to the amplitude of the filtered signal.

In [2], the MARGIN protocol is evaluated on three tasks: showing which parts of an image influence the most the prediction made by a DNN, pointing out to the hardest training examples to classify and detecting adversarial attacks. Experiments are performed using a common dataset of image classification and the Alexnet [29] DNN architecture.

MARGIN enables to obtain sparse saliency maps highlighting the importance of some group of pixels in the classification decision. MARGIN also enables to find hard examples. In a two-way classification task, trying to distinguish tabby cat and great dane, the images with the highest influence values (harder to classify) are the ones where the animal face is not visible. MARGIN is also useful to detect adversarial examples. An experiment shows that MARGIN induces a very sharp difference between the distribution of the adversarial examples and the one of the training examples, with only a small overlap in the distributions. These overlaps have been investigated and it was shown that they correspond to samples that are similar to the confusing examples from the previous experiment.

1.3.2   Graphs to Denoise Intermediate Representations

In this section, we introduce a few examples of methods based on GSP and aiming at "denoising" the features extracted from DNNs.

Graph filtering for localizing images

Visual-based localization (VBL) is the problem of retrieving the location and orientation (pose) of the camera which generated a given query image. For example, from one photograph of a place, the goal is to retrieve its GPS coordinates. Many different photographs can be taken from one place, with different angles of view, various weathers, different people on them. Thus, the VBL problem becomes challenging because of the difficulty to learn a representation which is resilient this amount of appearance variations.

Some researchers [6, 25] proposed to directly train a DNN to map images to GPS positions [6, 25]. However, such method comes with serious limitations. For example, they are unable to generalize to GPS positions unseen during training. Also, appending new GPS positions to the dataset requires retraining the whole DNN. Furthermore, small differences between the training images and a new image can cause significant localization errors.

In this subsection, another method addressing both limitations is presented [31]. It is based on Graph Signal Processing, and aims at improving the performance of VBL by incorporating additional available information. The additional information acts on the representations of the data, making them smoother on a graph designed using all available information. The consequence is a boost in localization performance.

The method requires both a support set of images from a predefined set of places and the access to a DNN trained to map images to representations that are more resilient to appearance changes [3]. The method does not require additional learning, allowing it to be possibly executed on a resource constrained system. Interestingly, it can be extended to new places without needing to retrain as well.

In details, the proposed method consists in denoising the representations exploiting a low-pass

graph filter [3, 46]. In the graph $\mathcal{G}$, each vertex is associated with an image. The edges model relations between images and are derived from the additional source of information (e.g. duration between two consecutive photographs, GPS positions, similarities of the representations...). In the following, the inference of the graph $\mathcal{G}$ and the graph filter are defined more precisely.

The graph $\mathcal{G}$ is described by its adjacency matrix $\boldsymbol{A}$ built from three different sources of information:

$$\mathbf{A} = \mathbf{A}_{\mathtt{dist}} + \mathbf{A}_{\mathtt{seq}} + \mathbf{A}_{\mathtt{sim}}, \tag{1.10}$$

where $\mathbf{A}_{\mathtt{dist}}$ represents the distance measured by the GPS coordinates between two vertices, $\mathbf{A}_{\mathtt{seq}}$ represents the delay in time acquisition between two images (acquired as frames in videos), $\mathbf{A}_{\mathtt{sim}}$ represents the cosine similarity between the representations of the two images.

The matrix $\mathbf{A}_{\mathtt{dist}}$ is parametrized by a scalar γ and a threshold parameter $\mathtt{dist}_{\max}$ cutting edges between distant vertices:

$$\mathbf{A}_{\mathtt{dist}}[i,j] = \begin{cases} e^{-\gamma \mathtt{dist}_{i,j}} & \text{if } \mathtt{dist}_{i,j} < \mathtt{dist}_{\max} \\ 0 & \text{otherwise} \end{cases}. \tag{1.11}$$

To exploit the information of time acquisition of frames, the function $seq(k, i, j)$, returns 1 if the frame distance between i and j is exactly k and 0 otherwise. The matrix $\mathbf{A}_{\mathtt{seq}}$ is parametrized by scalars β_k and k_{max}:

$$\mathbf{A}_{\mathtt{seq}}[i,j] = \sum_{k=1}^{k_{max}} \beta_k \mathtt{seq}(k, i, j). \tag{1.12}$$

The matrix $\mathbf{A}_{\mathtt{sim}}$ depends on the cosine similarity $sim_{\cos}$ between the representations of the images. It is parametrized by a scalar α, which controls the importance of the similarity with respect to the two other sources of information. The cosine similarity between two images is only considered if their GPS positions are close or if the photographs have been taken successively:

$$\mathbf{A}_{\mathtt{sim}}[i,j] = \begin{cases} \alpha sim_{\cos}(i,j) & \text{if } \mathbf{A}_{\mathtt{dist}}[i,j] > 0 \\ & \text{or } \mathbf{A}_{\mathtt{seq}}[i,j] > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{1.13}$$

Given the signal $\boldsymbol{s}$, the normalized Laplacian matrix $\boldsymbol{L}_{\mathrm{norm}} = \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{L} \boldsymbol{D}^{-\frac{1}{2}}$ and two hyperparameters a and m, the filtered signal is defined by the equation:

$$\tilde{\boldsymbol{s}} = (\mathbf{I} - a\boldsymbol{L})^m \boldsymbol{s}. \tag{1.14}$$

Note that when $m = 0$ no filtering is performed.

The authors stressed their method using two datasets collected in Australian cities (Adelaide and Sidney). The datasets are composed of videos taken by vehicles while they were moving around the cities. The goal is to learn to retrieve the GPS position of each frame.

The experiments show that on the Adelaide dataset, denoising increases performance, even when applied only on the test set, and as expected, using graph filters on both test and support gives the

best results. Second, on the Sydney dataset, while using the parameters optimized for the Adelaide dataset, the graph filter gets better performance in both median distance and accuracy. This shows that it is not needed to re-estimate parameters for a new dataset.

Manifold denoising for few-label classification

In classification, training a DNN requires a huge number of labeled data samples. When a DNN is trained on a small set of samples, it is likely to learn many irrelevant details of the data. It will not infer well the class of new samples coming to the classifier. This is a well known problem called *overfitting* (see Section 1.2.1). To handle overfitting, one can take advantage of the fact that DNNs are really good at extracting relevant information from data. Indeed, as a reminder a DNN can often be split into two functions $f = g \circ h$. The first part h learns to represent the data samples in a space where a simple classifier g can separate the data samples according to their classes. For example, when dealing with images, it is common to choose h as a convolutional neural network and g as a logistic regression.

The features of the data samples at the output space of the function h are often assumed to be only in a smooth subspace (manifold) of the entire output space. Sometimes, because of noise within the representations, features may appear outside this manifold. Removing the noise from the representations could help in making the features closer to their underlying structure, and thus, improve the performance.

The denoising method presented in this subsection is used in a few-label setting in which few labeled samples $\boldsymbol{X}_S$ and many unlabeled samples $\boldsymbol{X}_U$ are given. Consider a DNN, trained on a generic dataset, used to extract some features for a new few-shot classification task. A classifier will be trained on top of the extracted features to distinguish the new classes. However, the features may not be completely relevant for the new classification task. One may act directly on the training of the DNN using self-supervision, manifold-mixup or even training on a variety of side tasks [42, 43]. However, here, it is assumed that the features of all data samples are already given. In this case, one can exploit the additional information given by the representations of the unlabeled samples (more representative feature space). That is the case of a method based on GSP (Transfer+SGC [18]) that adds a simplified graph convolution (SGC) [60] on top of the extracted features. Note that SGC can be seen as a graph filter (previously defined in Section 1.2.2).

The authors use the notion of cosine similarity between the representations of two data samples in the feature space. Note that in the experiments, the features are always extracted after a ReLU function, so that they contain non-negative values. Thus, the output of the cosine similarity ranges from 0 to 1.

The idea is to build a k-nearest neighbor similarity graph $\mathcal{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, where each node represents a data sample. The connections between the nodes are weighted by the cosine similarity of their associated samples.

After removing self-loops, only the k most similar neighbors are kept for each vertex. Then, the resulting adjacency matrix $\mathbf{A}$ is normalized using the inverse square root of the degree matrix:

$$\boldsymbol{E} = \boldsymbol{D}^{-1/2} \mathbf{A} \boldsymbol{D}^{-1/2}.$$

Given $\boldsymbol{F}$ the matrix containing the features of all data samples and $\mathbf{I}$ the identity matrix, the new features are obtained by propagating the extracted features as follows:

$$\mathbf{F}_{\text{diffused}} = (\alpha \mathbf{I} + \boldsymbol{E})^m \mathbf{F} , \tag{1.15}$$

where α, m and k are hyperparameters. On top of the diffused features of the labeled samples, a logistic regression is trained to distinguish the labels.

The authors of [18] observed that their method enabled a significant gain in the 1-shot case and a smaller one in 5-shot with respect to existing alternatives. As a matter of fact, the added information coming from the unlabelled samples is relatively more important in the 1-shot case than it is for the 5-shot case.

Manifold denoising for semi-supervised classification

Here, a different scenario is considered in which a DNN $f = g \circ h$ is learned from scratch. It takes into input an image and outputs its class without overfitting on the training samples $\boldsymbol{X}_S$. However, in many cases, the training data are still not representative of all the possible data in the real world. In a semi-supervised setting, one solution to better fit the real data distribution is to use the unlabeled samples $\boldsymbol{X}_U$ as additional training data for the DNN. In the following paragraphs, an approach described in [21] will be presented. This approach exploits the so-called "manifold assumption": similar examples in the input space of the classifier g should have the same labels. This assumption is applied to label the unlabelled examples, so that they can be used as additional training data.

The method consists in merging the label propagation method, which is a classical semi-supervised learning method, with a DNN. First, a DNN is trained on the labeled examples to minimize a classification loss defined on the labeled images as:

$$L_s(X_S, Y_S, \theta) = \sum_{i=1}^{l} l_s(h(\boldsymbol{x}_i, y_i) , \tag{1.16}$$

where l_s is the cross-entropy loss.

A nearest-neighbor similarity graph is built at the input space of the classifier g. More precisely, each node of the graph corresponds to an image (labeled or unlabeled). The adjacency matrix $\mathbf{A}$ of the graph is defined so that $\mathbf{A}_{i,j}$ measures the similarity between $\boldsymbol{f}_i = h(\boldsymbol{x}_i)$ and $\boldsymbol{f}_j = h(\boldsymbol{x}_j)$. After removing self-loops ($\mathbf{A}_{i,i} = 0$), only the k-th largest values on each row of $\mathbf{A}$ are kept. To make $\mathbf{A}$ symmetric, its transposed version is added to it. As in the previous paragraphs, a normalized version of the adjacency matrix is used.

The core idea of the approach is to propagate the known labels on the similarity graph to define pseudo-labels for the unlabeled images. Indeed, a label matrix $\boldsymbol{Y} \in \mathbb{R}^{n \times c}$ is defined as:

$$\boldsymbol{Y}_{i,j} = \begin{cases} 1 & \text{if } 1 \le i \le l \text{ and } y_i = j \\ 0 & \text{otherwise} \end{cases} . \tag{1.17}$$

The pseudo-labels are obtained by diffusing the label matrix $\boldsymbol{Y}$ according to $\mathbf{A}$ until convergence (theoretical details in [64] and practical implementation in [21]):

$$\boldsymbol{Z} = (\mathbf{I} - \alpha\mathbf{A})^{-1}\boldsymbol{Y} \, , \tag{1.18}$$

where α is a hyperparameter. The pseudo-label $\hat{y}_i$ of an unlabeled sample x_i is finally defined as:

$$\hat{y}_i = \arg\max_j \boldsymbol{z}_{i,j} \, . \tag{1.19}$$

The set of pseudo-labels attributed to the unlabelled samples is noted $\hat{Y}_U$. A weight is associated with each pseudo-label to measure its uncertainty. Finally, the DNN is fine-tuned for a few epochs to minimize a new weighted loss that considers the pseudo-labels. The new loss is defined as:

$$L_w(X_S, Y_S, X_U, \hat{Y}_U, \theta) = \sum_{i=1}^{l} \zeta_{y_i} l_s(h(\boldsymbol{x}_i, y_i) + \sum_{i=l+1}^{n} \omega_i \zeta_{\hat{y}_i} l_s(h(\boldsymbol{x}_i, \hat{y}_i) \, , \tag{1.20}$$

where ζ_j is the reciprocal of the number of samples assigned to class j (known labels and pseudo labels).

The authors of [21] perform experiments using the CIFAR-10 and mini-Imagenet datasets. Both datasets are split into a training and a test set. In the training set, a number of images are labeled, the others are not.

In the semi-supervised setting, the error is reduced with respect to the fully-supervised setting. Without any surprise, the gain is observed to be decreasing when the number of labeled data points increases. To summarize, in this part it was described how to denoise the features of the data samples in three applications, where the graph filtering of either features or labels is considered. In the features case, it is assumed that the representations of all data samples should live on a manifold, and that the extracted features are noisy versions of these "ideal" representations. So, using a graph framework, it is possible to average the representation of the images with the most similar representations obtained with other images. In the second case, images having similar representations are assumed to have similar labels. Thus, the main goal is to smooth the label signal on the similarity graph whose vertices are the images and the edge weights are the similarity between the representations of two images.

1.3.3   Graphs as Losses

Training a DNN for classification often requires to optimize it in order to minimize a cross-entropy loss. As stated in Section 1.2.1, the cross-entropy loss introduces several biases in the learning process. This section gathers several alternative graph-based losses showing interesting properties.

Smoothness-based loss

In this subsection, a graph-based loss handling the limitations of the very popular cross-entropy loss is presented. Recall that the cross-entropy loss requires the outputs of the network to converge to label indicator vectors, which comes with several shortcomings described in Section 1.2.1.

Authors have proposed solutions to overcome these drawbacks. For example, in [17], the authors use triplets as loss instead of the cross-entropy. In a triplet, the first element is the example to train,

the second belongs to the same class and the last to another class. They update the parameters of the DNN so that the first example is closer to the second example than to the last one. Here, a similar loss based on GSP is presented. Contrary to the previous loss, the so-called graph smoothness loss only aims at maximizing the distances between outputs of different classes [4]. To avoid the DNN converging to a trivial solution that would scatter the outputs far away from each other regardless of their class, the outputs are constrained to remain in a compact subset of the output space.

Consider the problem of training a DNN f to classify examples. The input of the ith example is denoted $\boldsymbol{x}_i$ and the output of the DNN $\hat{\boldsymbol{y}}_i$, such that $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i)$. Given a distance metric $\|\cdot\|$ and a parameter α, a graph $\mathcal{G}$, whose vertices are examples, is generated. Its edges correspond to the similarity between the outputs of the examples. Thus, the adjacency matrix $\mathbf{A}$ of $\mathcal{G}$ is given by:

$$A_{i,j} \neq 0 \Rightarrow A_{i,j} = \exp\left(-\alpha \| f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) \|\right), \forall i, \forall j \ . \tag{1.21}$$

The matrix $\mathbf{A}$ is then thresholded in order to only keep the k biggest edges between the vertices. Finally, to store the information about a label c, a signal on $\mathcal{G}$ is defined as a binary indicator vector $\boldsymbol{s}_c \in \mathbb{R}^{|\mathbb{V}|}$. Hence, $s_c[i] = 1$ if and only if $\boldsymbol{x}_i$ has c as label.

According to the definition of the smoothness in Section 1.2.2, the graph smoothness loss over the graph $\mathcal{G}$ is given by:

$$
\begin{aligned}
\mathcal{L}_{\mathcal{G}} &= \sum_{\forall c} \boldsymbol{s}_c^\top \boldsymbol{L} \boldsymbol{s}_c \\
&= \underbrace{\sum_{\substack{\boldsymbol{x}_i, \boldsymbol{x}_j, W_k[i,j] \neq 0 \\ s_c[i] s_c[j] = 0, \forall c}}}_{\text{sum over inputs of distinct classes}} \exp\left(-\alpha \| f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) \|\right) \ .
\end{aligned}
$$

The graph smoothness loss addresses the following drawbacks of the cross-entropy loss:

- The dimension of the DNN output is less tightly tied to the number of classes with the proposed loss than with the cross-entropy one.

- Keeping only k edges per vertex gives more flexibility to the proposed loss: using a small value of k, it is possible to minimize the graph smoothness loss with multiple clusters of points for each class.

- The graph-smoothness loss is only interested in relative positioning of outputs with regards to one another, and is therefore built upon the initial distribution yielded by the network.

In [4], the performance of the graph-smoothness loss is evaluated using three common datasets of image classification. For each dataset, a Resnet-18 [15] is trained with the cross-entropy loss and another one with the graph smoothness one. To perform classification with the graph smoothness loss, an additional classifier (in the paper a K-nearest neighbors classifier with $K = 10$) is also trained on top of the DNN.

The authors show that the graph smoothness loss is able to compete in terms of raw performance with the cross-entropy one. They also perform robustness experiments where they observe that the networks trained with the graph smoothness loss are able to better handle additive noise to the processed inputs.

Graph knowledge distillation

To achieve state of the art results, DNNs often rely on a large number of trainable parameters, and considerable computational complexity. This is why there has been a lot of interest in the past few years towards their compression, so that they can be deployed onto embedded systems or in real-time settings. The purpose of this subsection is to analyze a graph-based technique, called Graph Knowledge Distillation (GKD), that allows for an efficient compression of the DNN architecture, while maintaining a high level of accuracy. This framework has been already used with success in the literature in [33, 36, 37].

Distillation-based approaches aim at distilling knowledge from a pre-trained larger network that is called teacher to a smaller yet to be trained network called student. For simplicity, assume that both architectures always generate the same number of intermediate representations, even if they do not have the same depth. The loss function of the student network trained with knowledge distillation is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{KD}} \cdot \mathcal{L}_{\text{KD}} , \tag{1.22}$$

where $\mathcal{L}_{\text{task}}$ is the same loss as the one used to train the teacher (e.g. cross-entropy), $\mathcal{L}_{\text{KD}}$ is the distillation loss and λ_{KD} is a scaling parameter to control the importance of the distillation with respect to that of the task.

Given an architecture A, a batch of inputs $\boldsymbol{X}$ and a subset of layers, the set $\mathbb{X}'$ contains the intermediate representations of $\boldsymbol{X}$ after all considered layers. For each layer, the representations after the layer $\boldsymbol{X}' \in \mathbb{X}'$ are used to define a similarity graph $\mathcal{G}^A(\boldsymbol{X}')$. These graphs contain a vertex for each input in the batch, and the weight of the edge between two vertices is inferred using a measure of similarity between the representations of the associated inputs. Finally, in order to control the importance of outliers, the adjacency matrices of the graphs are normalized ($\mathbf{A} = \boldsymbol{D}^{-\frac{1}{2}} \mathbf{A} \boldsymbol{D}^{-\frac{1}{2}}$).

While training the student, the training batch of inputs goes through both the student architecture and the (now fixed) previously trained teacher architecture. The loss to minimize combines the task loss, as expressed in Equation 1.22, with the following graph knowledge distillation (GKD) loss:

$$\mathcal{L}_{\text{KD}} = \sum_{\boldsymbol{X}' \in \mathbb{X}'} \mathcal{L}_d(\mathcal{G}^S(\boldsymbol{X}'), \mathcal{G}^T(\boldsymbol{X}')) , \tag{1.23}$$

where $\mathcal{L}_d$ is the squared Frobenius norm between the adjacency matrices. The GKD loss measures the discrepancy between the adjacency matrices of teacher and student graphs. In this way, the geometry of the intermediate representations of the student is forced to converge to the one of the teacher. The intuition is that since the teacher network is expected to generalize well to the test, mimicking the geometry of its intermediate representations should allow the student network to generalize better. An equivalent definition of the proposed loss is:

$$\mathcal{L}_{\text{KD}} = \sum_{\boldsymbol{X}' \in \mathbb{X}'} \|\mathbf{A}^S(\boldsymbol{X}') - \mathbf{A}^T(\boldsymbol{X}')\|_2^2 . \tag{1.24}$$

In [33], the advantage of using GKD compared with a baseline method (RKD-D [45]) is evaluated using CIFAR-10 and CIFAR-100 datasets [28]. The obtained results are analyzed on their: i) accuracies, ii) decision consistency, and iii) spectral representations.

In terms of accuracy, the authors compare student sized networks trained without distillation (Baseline), with GKD and RKD-D GKD methods. The comparison shows that RKD-D by itself provides a small gain in error rate with respect to the Baseline approach, while GKD outperforms RKD-D by almost a similar amount. Furthermore, the performance of GKD are shown to be better when using task specific graphs. In other words, by removing the edges between elements of the same class. GKD with task specific graphs provides a gain over GKD of the same magnitude as GKD over RKD-D. Thus, by going from RKD-D to GKD-task specific it was possible obtain a twofold gain over the baseline.

In terms of decision consistency, the authors evaluate the consistency by taking the trained students and comparing their outputs to the trained teacher's outputs. It was reported that the ideal scenario that greatly improves the classification performance occurs when the student is 100% consistent with the teacher's decision on the test set. The experiment shows that the GKD is more consistent with the teacher than the RKD-D.

In terms of spectral representations, it is quite natural to analyze performance from a GSP perspective [50]. Considering specific graph signals, the respective smoothness on each of the two graphs (RKD-D and GKD) are compared. Two signals are considered: i) the label binary indicator signal, and ii) the Fiedler eigenvectors from each intermediate representation in the teacher. Experiments show that both signals are smoother in the networks trained with GKD. This means that the geometry of the intermediate spaces from GKD are more aligned to those of the teacher.

Affinity-based loss

In this subsection, we present another way of defining the loss of a DNN using graphs [59]. During training, a DNN is evaluated on a whole batch of data samples before being updated. The idea described in [59] is to ease the training by exploiting meaningful relationships between these data samples. In short, the idea amounts to add a regularization term to the usual loss of the DNN. While the usual loss is only focused on the task to achieve, the regularization term is more representative of the relationships between the data samples within the DNN.

Let us consider that a DNN is trained on a batch of N examples. The input of the ith sample is denoted $\boldsymbol{x}_i$. The representation of $\boldsymbol{x}_i$ obtained after a given layer of the DNN is denoted $f(\boldsymbol{x}_i)$. Depending on the task, f can represent a hidden layer or the output of the DNN.

The affinity between two samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ within the DNN can be computed with any similarity function $\mathcal{W}$, as

$$A_{i,j} = \mathcal{W}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)). \tag{1.25}$$

Then, to model the relationships between the examples within the DNN, a graph $\mathcal{G}$ is defined. Its vertices are the examples. The connection between two examples i and j is weighted by their affinity $A_{i,j}$. The adjacency matrix of this affinity graph is denoted by $\mathbf{A}$.

Now, the idea in [59] is to select a meaningful set of connections that we will aim at maximizing during training. For instance, in a classification task, we want to maximize the similarity between the outputs of data samples of the same class. So, the set of meaningful connections amounts to the set of

connections between the examples of the same class. Formally, to select the relevant connections, we need to define a target matrix $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{N \times N}$. Given S the set of meaningful pairwise connections, the target $\boldsymbol{\mathcal{T}}$ is defined as

$$\mathcal{T}_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in S, \\ 0, & \text{otherwise.} \end{cases}$$

Finally, given $\hat{\mathbf{A}} = \text{softmax}(\mathbf{A})$ a matrix-wise softmax operation, the quantity we aim at maximizing is:

$$\mathcal{M} = \sum \hat{\mathbf{A}} \odot \boldsymbol{\mathcal{T}}.$$

In [59], the maximization of $\mathcal{M}$ is expressed as the minimization of a loss $\mathcal{L}_{\mathcal{G}}$. A discussion on the choice of the loss form is made in the original paper [59].

Finally, the total loss of the DNN is given by

$$\mathcal{L} = \mathcal{L}_{task} + \lambda \mathcal{L}_{\mathcal{G}},$$

where $\mathcal{L}_{task}$ could be the cross-entropy loss, and λ is a parameter that takes values in $[0,1]$. The minimization of the affinity loss $\mathcal{L}_{\mathcal{G}}$ places greater emphasis on the connections in S during training.

In [59], experiments are made on CIFAR-10, CIFAR-100 and Tiny-ImageNet are summarized. The experiments show that there is an improvement in terms of classification accuracy over a baseline, when using the affinity-based loss. In particular, for datasets with a large number of categories, such as CIFAR-100 (100 classes) and Tini-ImageNet (200 classes), the performance gain is above 1%.

1.3.4   Graphs as Regularizers

In this section, a graph-based technique is used to improve the robustness of DNNs to deviations of the inputs. In other words, the goal is to ensure that a small difference in the input does not lead to a large difference in the output (decision) of the DNNs.

Bounding smoothness evolution between layers

In this subsection, a regularizer that penalizes large deformations of the class boundaries (i.e. distances between the closest elements of distinct classes) throughout the network architecture is presented [32]. The regularizer is defined independently of the types of perturbations that is expected to face when the system is deployed.

The regularizer is based on a series of graphs, one for each layer of the DNN. Each graph captures the similarity between the intermediate representations of training examples at that layer.

Given a batch of b inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_b\}$ and a similarity measure sim, the adjacency matrix of the graph generated by intermediate representations at layer ℓ is defined as:

$$\mathcal{A}_{i,j}^{\ell} = \text{sim}(\boldsymbol{x}_i^{\ell+1}, \boldsymbol{x}_j^{\ell+1}), \forall 1 \le i, j \le b. \tag{1.26}$$

In the experiments, the cosine similarity is mainly used.[2]

To measure the deformation induced by a given layer ℓ, a regularizer δ_σ^ℓ is computed as the difference between label signal smoothness before and after the layer for all labels. More precisely,

$$\delta_\sigma^\ell = \sum_c \left| \sigma^\ell(\mathbf{s}_c) - \sigma^{\ell-1}(\mathbf{s}_c) \right|, \tag{1.27}$$

where σ^ℓ is the smoothness (Section 1.2.2) of the binary label indicator vector on the intermediate representation graph from layer ℓ. These quantities are then used to regularize modifications made by each of the layers during the learning process.

The regularizer penalizes large changes in the smoothness (computed using the Laplacian quadratic form) of the class indicator vectors (viewed as "graph signals") between layers. As a result, the margin is kept almost constant across layers, and the deformations of space are controlled at the boundary regions. More details can be found in the article [34].

Since only label signals are considered, only the similarities between examples of distinct classes are of interest. As such, the regularizer only focuses on the boundary, and does not vary if the distance between examples of the same label grows or shrinks.

The experiments show that the regularizer is able to increase robustness to random perturbations and to weak adversarial attacks on the CIFAR-10 dataset. To validate the generality of the method, the reader is referred to [34, 35], where the analysis is extended to two other datasets (CIFAR-100 [28] and Imagenet32x32 [8]). In the following, the regularizer robustness to some common perturbations is discussed: i) noise [16, 40], ii) adversarial attacks [10, 54], and iii) implementation defects.

The robustness of the network to noise is evaluated. The benchmark from [16] is used. The benchmark consists in 15 different perturbations, with 5 levels of severity each. it is observed that the regularizer is able to increase the relative performance under perturbations by a significant amount (using the mREI measure from [34] an improvement of 0.29 is noticed).

The robustness to adversarial inputs is evaluated, which are specifically built to fool the DNN. The network performance is evaluated by adding the scaled gradient sign (FGSM attack) to the input [30], so that a target SNR of 33 is obtained. The experiments show that the regularizer increases robustness against adversarial attacks as the regularizer improves the network's error rate from 66% to 49%.

The robustness of the architecture to noise on parameters and activations is evaluated. Two types of noises are considered: i) erasures of the memory (dropout), and ii) quantization of the weights [19].

Both experiments show that the regularizer is able to improve the robustness. In the case of the dropout experiment, the regularizer improves the error rate from 55% to 34% and in the case of quantization the error rate improves from 76% to 50%.

---

[2]It is often the case that the output $\boldsymbol{x}^{\ell+1}$ is obtained right after using a ReLU function, that forces all its values to be nonnegative, so that all values in $\mathbf{A}^\ell$ computed with the cosine similarity are between 0 and 1.

Peernets: mixing graph and image filters to improve robustness

In this subsection a second graph-based method which specifically addresses the problem of adversarial attacks [52] is presented. Small perturbations on an input image, often invisible for the human eye, can mislead a DNN and completely distort its prediction. The authors of [52] propose to modify the architecture of standard DNNs to make DNNs more robust to adversarial attacks.

The intuition was that smoothing some fragments of the intermediate representations of the images with a linear combination of fragments from intermediate representations of other similar images would improve the robustness without damaging too much the accuracy. In this way, this should allow one to "filter away" (with a graph filter) the adversarial noise.

The authors of [52] propose to implement this concept as a new layer, called the *Peer Regularization layer*, to intertwine within standard layers of existing DNN architectures. They have been inspired from various domains including Deep learning on graphs, low-dimensional regularization and non-local image filtering. In the following, a detailed description of the Peer Regularization layer is given.

Consider N images (the peers) and their intermediate representations at the output of a given layer. The representation of an image is a $n \times d$ matrix, where n is the number of pixels and d is the number of features in each pixel.

The idea of the Peer Regularization layer is to leverage the structure of the feature space of the intermediate representations obtained after a conventional layer (e.g. after a CNN). Each pixel of each intermediate representation is replaced by a smoothed version of itself. The smoothed version is computed from the K most similar pixels among all the pixels belonging to the intermediate representations of all the other images (the peers). More precisely, for each pixel of each intermediate representation, a K nearest neighbor graph is computed. This means that the similarities (for instance, cosine similarities) are computed between the d-dimensional vector representing the pixel and all d-dimensional vectors representing pixels of all the peer images. Then, only the K pixels showing the biggest similarities are kept to compute the smoothed version of the initial pixel.

The d-dimensional vectors of the K pixels are combined using an attention mechanism. Let us denote the pth pixel of image i $\boldsymbol{x}_p^i$. Its kth neighbor is a pixel q_k from an image j_k written $\boldsymbol{x}_{q_k}^{j_k}$. The smoothed version of the pixel $\tilde{\boldsymbol{x}}_p^i$ is given by the equation:

$$\tilde{\boldsymbol{x}}_p^i = \sum_{k=1}^{K} \alpha_{ij_kpq_k} \boldsymbol{x}_{q_k}^{j_k}, \qquad \alpha_{ij_kpq_k} = \frac{\text{LeakyReLU}(\exp(a(\boldsymbol{x}_p^i, \boldsymbol{x}_{p_k}^{j_k})))}{\sum_{k'=1}^{K} \text{LeakyReLU}(\exp(a(\boldsymbol{x}_p^i, \boldsymbol{x}_{p_{k'}}^{j_{k'}})))} \ . \tag{1.28}$$

The function a is a fully connected layer whose input is a $2d$-dimensional vector and outputs a scalar. The α represents the relative importance of each neighboring pixel.

In the original paper, the authors report experiments on CIFAR-10 with a Resnet-32 containing intertwined Peer Regularization layers (PR-Resnet-32). The test images are disrupted with three white-box adversarial attacks (gradient descent, fast-gradient sign method, projected gradient descent) of varying amplitude (measured by the variable ϵ). It was shown that PR-Resnet-32 is more robust under these adversarial attacks than the standard Resnet-32 (more details in the original paper [52]).

While this kind of methods is not perfect yet, it shows an interesting example of how graph

methods can help to handle a Deep Learning problem. Here, the graph enables to take advantage of the relationships between data at any point within a DNN, and to use this knowledge to make it more robust to small perturbations. Note however, that it requires additional training time and arbitrary choices about the training images to keep during testing. More information on the implementation can be found in the original paper.

1.4 Conclusion

DNNs achieve state of the art performance in many machine learning tasks. Unfortunately, their practical uses in real-world applications are still limited due to some major limitations. In this chapter, we have presented an overview of graph-based tools that aim at solving some of these limitations. Namely, i) interpreting the intermediate representations of DNNs, ii) train models with few data, iii) learning meaningful embeddings for classification, iv) improving the robustness of DNNs, and v) compressing the architectures.

Approaching these limitations with graphs enables to view them from a new angle. In many methods, the graphs are used to model the intermediate spaces of the DNNs. In general, their vertices represent distinct data samples and their edges depend on the similarity of the intermediate representations of the data samples within the DNNs. Such graphs are called *latent geometry graphs*. Given the ubiquity of graphs in representing relations between data samples, it is quite natural to imagine that GSP, a branch of the signal processing field that handles data supported on graphs, is a valuable path to address these challenges.

This area of research is in its infancy. We hope that such an overview will inspire researchers to work in this line of research that we believe could help improving the applications of deep learning that we see in day-to-day life.

Bibliography

[1] Anirudh, R., Bremer, P., Sridhar, R., and Thiagarajan, J. (2017a). Influential sample selection: A graph signal processing approach. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States). 6, 13

[2] Anirudh, R., Thiagarajan, J. J., Sridhar, R., and Bremer, T. (2017b). Margin: Uncovering deep neural networks using graph signal analysis. *arXiv preprint arXiv:1711.05407*. 14

[3] Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T., and Sivic, J. (2016). Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307. 6, 14, 15

[4] Bontonou*, M., Lassance*, C., Hacene, G. B., Gripon, V., Tang, J., and Ortega, A. (2019). Introducing graph smoothness loss for training deep learning architectures. In *2019 IEEE Data Science Workshop (DSW)*, pages 160–164, * authors contributed equally. 6, 19

[5] Bontonou, M., Béthune, L., and Gripon, V. (2020). Predicting the accuracy of a few-shot classifier. *arXiv preprint arXiv:2007.04238*. 6

[6] Brahmbhatt, S., Gu, J., Kim, K., Hays, J., and Kautz, J. (2018). Geometry-aware learning of maps for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625. 14

[7] Burt, J. R., Torosdagli, N., Khosravan, N., RaviPrakash, H., Mortazi, A., Tissavirasingham, F., Hussein, S., and Bagci, U. (2018). Deep learning beyond cats and dogs: recent advances in diagnosing breast cancer with deep neural networks. *The British journal of radiology*, **91**(1089), 20170545. 3

[8] Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*. 23

[9] Defferrard, M., Martin, L., Pena, R., and Perraudin, N. (2017). Pygsp: Graph signal processing in python. *Zenodo*. 10

[10] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*. 6, 7, 23

[11] Gripon, V., Ortega, A., and Girault, B. (2018). An inside look at deep neural networks using graph signal processing. In *Proceedings of ITA*. 6, 12, 13

[12] Gupta, V., Sambyal, N., Sharma, A., and Kumar, P. (2019). Restoration of artwork using deep neural networks. *Evolving Systems*, pages 1–8. 3

[13] Hacene, G. B. (2019). *Processing and learning deep neural networks on chip*. Ph.D. thesis, Ecole nationale supérieure Mines-Télécom Atlantique. 3

[14] Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, **30**(2), 129–150. 9

[15] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. 3, 5, 19

[16] Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*. 7, 23

[17] Hermans, A., Beyer, L., and Leibe, B. (2017). In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*. 3, 6, 18

[18] Hu, Y., Gripon, V., and Pateux, S. (2020). Exploiting unsupervised inputs for accurate few-shot classification. *arXiv preprint arXiv:2001.09849*. 16, 17

[19] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, **18**(1), 6869–6898. 23

[20] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. 4

[21] Iscen, A., Tolias, G., Avrithis, Y., and Chum, O. (2019). Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 17, 18

[22] Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. (2019). Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*. 6

[23] Jiang*, Y., Neyshabur*, B., Mobahi, H., Krishnan, D., and Bengio, S. (2020). Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*. 6

[24] Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., *et al.* (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, **5**, 339–351. 3

[25] Kendall, A., Grimes, M., and Cipolla, R. (2015). Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946. 14

[26] Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille. 6

[27] Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2019). Large scale learning of general visual representations for transfer. 6

[28] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf*. 20, 23

[29] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 3, 14

[30] Kurakin, A., Goodfellow, I. J., and Bengio, S. (2017). Adversarial machine learning at scale. *International Conference on Learning Representations*. 23

[31] Lassance, C., Latif, Y., Garg, R., Gripon, V., and Reid, I. (2019). Improved visual localization via graph smoothing. *arXiv preprint arXiv:1911.02961*. 14

[32] Lassance, C., Gripon, V., Tang, J., and Ortega, A. (2019). Structural robustness for deep learning architectures. In *2019 IEEE Data Science Workshop (DSW)*, pages 125–129. 22

[33] Lassance, C., Bontonou, M., Hacene, G. B., Gripon, V., Tang, J., and Ortega, A. (2020). Deep geometric knowledge distillation with graphs. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8484–8488. IEEE. 20

[34] Lassance, C., Gripon, V., and Ortega, A. (Under review). Laplacian networks: Bounding indicator function smoothness for neural networks robustness. *APSIPA Transactions on Signal and Information Processing*. 23

[35] Lassance, C. E. R. K., Gripon, V., and Ortega, A. (2018). Laplacian networks: Bounding indicator function smoothness for neural network robustness. *arXiv preprint arXiv:1805.10133*. 23

[36] Lee, S. and Song, B. (2019). Graph-based knowledge distillation by multi-head attention network. *arXiv preprint arXiv:1907.02226*. 20

[37] Liu, Y., Cao, J., Li, B., Yuan, C., Hu, W., Li, Y., and Duan, Y. (2019). Knowledge distillation via instance relationship graph. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7096–7104. 20

[38] Ma, J., Zhou, C., Cui, P., Yang, H., and Zhu, W. (2019). Learning disentangled representations for recommendation. In *Advances in Neural Information Processing Systems*, pages 5711–5722. 3

[39] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*. 7

[40] Mallat, S. (2016). Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, **374**(2065), 20150203. 23

[41] Mangla, P., Singh, M., Sinha, A., Kumari, N., Balasubramanian, V. N., and Krishnamurthy, B. (2019). Charting the right manifold: Manifold mixup for few-shot learning. *CoRR*, **abs/1907.12087**. 6

[42] Mangla, P., Kumari, N., Sinha, A., Singh, M., Krishnamurthy, B., and Balasubramanian, V. N. (2020). Charting the right manifold: Manifold mixup for few-shot learning. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2218–2227. 16

[43] Milbich, T., Roth, K., Bharadhwaj, H., Sinha, S., Bengio, Y., Ommer, B., and Cohen, J. P. (2020). Diva: Diverse visual feature aggregation fordeep metric learning. *arXiv preprint arXiv:2004.13458*. 16

[44] Ortega, A., Frossard, P., Kovačević, J., Moura, J. M., and Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, **106**(5), 808–828. 9

[45] Park, W., Kim, D., Lu, Y., and Cho, M. (2019). Relational knowledge distillation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3967–3976. 20

[46] Radenović, F., Tolias, G., and Chum, O. (2019). Fine-tuning cnn image retrieval with no human annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **41**(7), 1655–1668. 15

[47] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. 6

[48] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., *et al.* (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, **115**(3), 211–252. 3

[49] Sandryhaila, A. and Moura, J. M. (2014). Discrete signal processing on graphs: Frequency analysis. *IEEE Transactions on Signal Processing*, **62**(12), 3042–3054. 11

[50] Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, **30**(3), 83–98. 4, 8, 9, 10, 21

[51] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., *et al.* (2017). Mastering the game of go without human knowledge. *Nature*, **550**(7676), 354. 3

[52] Svoboda, J., Masci, J., Monti, F., Bronstein, M., and Guibas, L. (2019). Peernets: Exploiting peer wisdom against adversarial attacks. In *International Conference on Learning Representations*. 24

[53] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014a). Intriguing properties of neural networks. In *International Conference on Learning Representations*. 4

[54] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014b). Intriguing properties of neural networks. In *International Conference on Learning Representations*. 6, 23

[55] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. (2018). A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer. 6

[56] Taylor, B. J. (2006). *Methods and procedures for the verification and validation of artificial neural networks*. Springer Science & Business Media. 4

[57] Tremblay, N., Gonçalves, P., and Borgnat, P. (2018). Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pages 299–324. Elsevier. 9

[58] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008. 3

[59] Wang, C., Samari, B., Kim, V. G., Chaudhuri, S., and Siddiqi, K. (2020). Affinity graph supervision for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8247–8255. 21, 22

[60] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International Conference on Machine Learning*. 16

[61] Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., and Stolcke, A. (2018). The microsoft 2017 conversational speech recognition system. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5934–5938. IEEE. 3

[62] Zhai, X., Puigcerver, J., Kolesnikov, A., Ruyssen, P., Riquelme, C., Lucic, M., Djolonga, J., Pinto, A. S., Neumann, M., Dosovitskiy, A., Beyer, L., Bachem, O., Tschannen, M., Michalski, M., Bousquet, O., Gelly, S., and Houlsby, N. (2019). A large-scale study of representation learning with the visual task adaptation benchmark. 6

[63] Zhao, Z.-Q., Zheng, P., Xu, S.-t., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, **30**(11), 3212–3232. 3

[64] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2004). Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328. 18