An Inside Look at Deep Neural Networks using Graph Signal Processing

Vincent Gripon[1], Antonio Ortega[2], and Benjamin Girault[2]

[1]IMT Atlantique, Brest, France
Email: vincent.gripon@imt-atlantique.fr

[2]University of Southern California, Los Angeles, CA
Email: firstname.lastname@usc.edu

Abstract—**Deep Neural Networks (DNNs) are state-of-the-art in many machine learning benchmarks. Understanding how they perform is a major open question. In this paper, we are interested in using graph signal processing to monitor the intermediate representations obtained in a simple DNN architecture. We compare different metrics and measures and show that smoothness of label signals on k-nearest neighbor graphs are a good candidate to interpret individual layers role in achieving good performance.**

I. Introduction

Deep Neural Networks (DNNs) have attracted a lot of interest for their ability to achieve state-of-the-art performance in numerous challenges in machine learning. It is thus clear that the mathematical functions that are obtained by assembling neural layers manage to greatly approximate associations between input natural signals and their corresponding labels. Among other explanations (e.g. [1]), there are two arguments that are of paramount importance: a) DNNs are universal approximators of continuous functions in vector spaces, and b) despite containing a huge number of parameters, they can efficiently be trained thanks to the chain rule of back propagation.

On the other hand, DNNs still suffer from major limitations:

- Since they typically rely on many parameters to reach good accuracy, they are not well fitted to learning challenges where a limited amount of data is available [2].
- Because they rely on gradient descent for training, DNNs are subject to catastrophic forgetting when training data is streamed [3].
- Avoiding overfitting boils down to performing crossvalidation, which requires very high quality representative training sets [4], [5].
- There is no systematic method today to efficiently find good hyperparameters for these architectures.

Most of these limitations come from the fact DNNs are in the end "black boxes". As such, there is a need for theories to explain their functioning.

In this paper we propose to use Graph Signal Processing (GSP) to monitor intermediate representations of DNNs. GSP is a framework that extends classical Fourier analysis to any topological domain that can be described by a graph. Our motivation is to use GSP to detect overfitting in a DNN architecture, and, more generally, to attempt at better understanding how intermediate representations of DNNs synergy to reach top performance.

The outline of the paper is as follows. In Section II we introduce DNNs and GSP. In Section III we introduce ways to characterize intermediate representations of DNNs using GSP. In Section IV we perform experiments on the CIFAR-10 dataset. Finally, Section V is a conclusion.

II. Deep Neural Networks and Graph Signal Processing

A. Deep Neural Networks

1) Classification: Classification is a common problem in the field of machine learning. It consists in finding a way to associate any observation x_{test} with a label y_{test}, generalizing previously given associations (x_{train}, y_{train}). It can thus be considered as a specific form of regression, where labels can typically only take a finite number of distinct values.

In many practical cases it is possible to represent both inputs and outputs as vectors (or tensors) with real valued coordinates.

2) Deep Neural Networks: DNNs are mathematical functions characterized by a large number of tunable parameters. They are built by assembling neural layers, which are typically made of two key ingredients: a linear transformation $\mathbf{W}^\ell$ and a nonlinear function h^ℓ, where ℓ indices the layer.

Denoting $\mathbf{x}^\ell$ and $\mathbf{y}^\ell$ resp. the input and the output of such a layer, a layer can be represented mathematically as follows:

$$\mathbf{y}^\ell = h^\ell(\mathbf{W}^\ell \mathbf{x}^\ell) \; .$$

In the most generic case where $\mathbf{W}^\ell$ is not constrained and h is a simple real valued function applied coefficientwise (typically a Rectified Linear Unit (ReLU): $f : x \mapsto \max(0, x)$), we term the layer "fully-connected". Of particular interest is a subfamily of layers where $\mathbf{W}^\ell$ is a tensor whose slices along its first dimensions are convolutional matrices. We call them "convolutional layers" (or "conv layers" for short). Note that it is known that convolutional matrices with a small localized kernel often lead to the best performance.

To the contrary, some layers are such that $\mathbf{W}^\ell$ contains almost no free parameters and h takes into account the specific underlying structure of $\mathbf{x}^\ell$. It is in particular the case for stride layers that consist of regularly sampling $\mathbf{x}$, or pooling layers in which a local filter is used to merge neighboring coordinates in $\mathbf{x}$.

These various layers can be assembled in many different ways, even though most existing architectures lead to underlying graphs that are mostly lines or trees. Most of the time the output of a layer is directly fed onto the input of a downstream layer. In some rare exceptions, concatenations or sums are used to merge the outputs of two or more layers.

3) Example architecture: In the experiments section, we make use of a specific architecture of a DNN that is summarized in Figure 1. This architecture takes as input tensors with dimensions 32x32x3 and outputs a 10-dimensional vector. We use small 3x3 kernels for convolutions, so that each convolutional layer has the effect of truncating a bit the first two dimensions of its input to avoid side-effects.

During training, we introduce dropout (i.e. random reset to 0 for some coordinates) with probability 0.1 independently for each coordinate right after Layers 3 and 6.

4) Training and testing: To train a DNN, we present the couples $(\mathbf{x}_{train}, \mathbf{y}_{train})$ to the network and perform a stochastic backpropagation of gradient descent. The parameters of the model are all free parameters in tensors $\mathbf{W}$, that are initialized at random. Unless stated otherwise, we use a RMSProp optimizer with a learning rate of 0.0001 and a decay of 1e-6. We call epoch a period of time during which all training vectors have been processed once. We call data augmentation adding artificial training examples that are small modifications of existing ones using translations, rotations and/or flipping.

Generally speaking, a key question to designing efficient DNNs resides in choosing good hyperparameters (e.g. number of layers, way of assembling them, number of free parameters in each layer, types on nonlinearities, optimization routine's choices...). Roughly speaking, there are two types of behavior that should be avoided: underfitting, which corresponds to the case where the architecture is not even able to obtain a good association in training pairs (x_{train}, y_{train}), and overfitting, which corresponds to the case where the architecture is obtaining performance significantly better on the training examples than on actual tests.

Crossvalidation is often used to estimate overfitting, and consists in splitting the training set in two parts: the first one is used for training and the second one for checking the ability of the architecture to generalize to previously unseen inputs.

However in many practical cases it is problematic to rely on crossvalidation, because:

- The training set is reduced in size. In some cases there are not enough training examples in total to even simply check the ability of the architecture to generalize efficiently,
- 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.

B. Graph Signal Processing

Graph signal processing [6] is a framework that aims at extending classical harmonic analysis to arbitrary domains

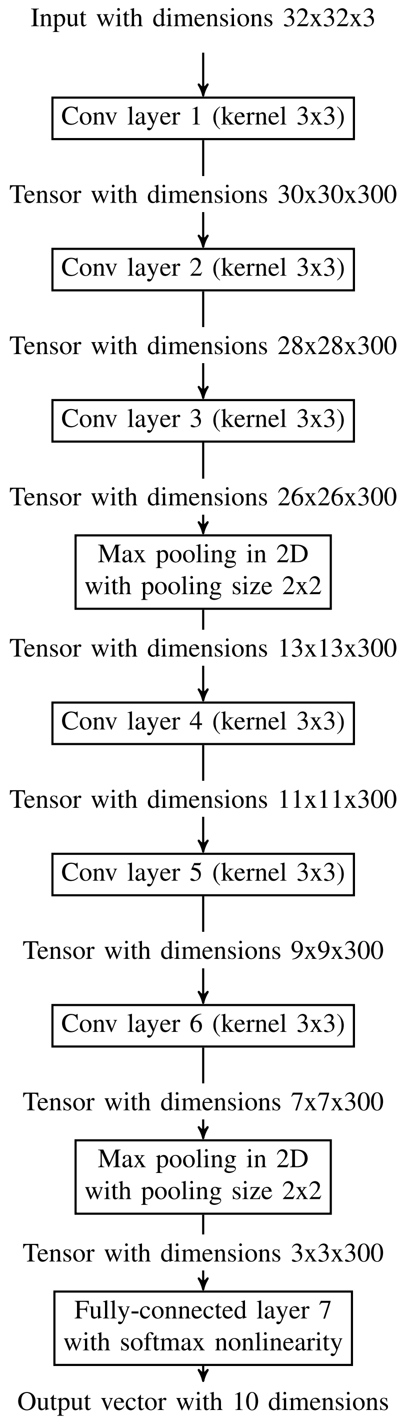

Input with dimensions 32x32x3
Conv layer 1 (kernel 3x3)
Tensor with dimensions 30x30x300
Conv layer 2 (kernel 3x3)
Tensor with dimensions 28x28x300
Conv layer 3 (kernel 3x3)
Tensor with dimensions 26x26x300
Max pooling in 2D with pooling size 2x2
Tensor with dimensions 13x13x300
Conv layer 4 (kernel 3x3)
Tensor with dimensions 11x11x300
Conv layer 5 (kernel 3x3)
Tensor with dimensions 9x9x300
Conv layer 6 (kernel 3x3)
Tensor with dimensions 7x7x300
Max pooling in 2D with pooling size 2x2
Tensor with dimensions 3x3x300
Fully-connected layer 7 with softmax nonlinearity
Output vector with 10 dimensions

Figure 1. Depiction of a DNN architecture designed to handle input images of 32x32x3 dimensions.

described by a graph. We call (weighted) graph a couple $G = \langle V, \mathbf{A} \rangle$ where V is a finite set of vertices indexed from 1 to n, the cardinality of V, and $\mathbf{A} \in \mathbb{R}^{n^2}$ is a symmetric matrix with nonnegative weights.

The (combinatorial) Laplacian of a graph G is the matrix $\mathbf{L} \triangleq \mathbf{D_A} - \mathbf{A}$, where $\mathbf{D_A}$ is the degree matrix of $\mathbf{A}$, that is to say the diagonal matrix which diagonal elements are the sum of corresponding rows in $\mathbf{A}$.

By construction, the Laplacian of a graph is a symmetric real-valued matrix, and can thus be decomposed as $\mathbf{L} = \mathbf{F}\Lambda\mathbf{F}^T$, where Λ is a diagonal matrix of eigenvalues on its diagonal in ascending order, $\mathbf{F}$ is an orthonormal matrix of eigenvectors and $\mathbf{F}^\top$ is its transpose. Note that the first column of F can always be chosen as a constant vector and is associated with eigenvalue 0. More generally, the multiplicity of eigenvalue 0 is the number of connected component in G. Note that a corollary of this proposition is that all eigenvalues are nonnegative.

Interestingly in the case of a ring graph columns of F can be chosen as classical discrete Fourier Modes. Following this lead, we define a signal $\mathbf{x}$ as a vector with dimension n, and $\hat{\mathbf{x}} \triangleq \mathbf{F}^\top\mathbf{x}$ as its Graph Fourier Transform (GFT). Reciprocally, $\mathbf{x}$ is said to be the inverse Graph Fourier Transform (iGFT) of $\hat{\mathbf{x}}$.

The smoothness of a signal on G is defined as the quantity $\mathbf{x}^\top L\mathbf{x} = \hat{\mathbf{x}}^\top\Lambda\hat{\mathbf{x}} = \sum_{i=1}^{n}\Lambda_{ii}\mathbf{x}_i^2$. Note that we choose to name this quantity "smoothness" to be coherent with existing literature, and despite the fact it would be more adequate to call it "nonsmoothness". Indeed, the smoothest signal on any graph is obtained when considering a constant vector, which by definition has a smoothness of 0. To the contrary, when considering unit-normed signals, the maximum value of smoothness is reached when considering the last column of F or its opposite, which typically involves a lot of zero-crossings (i.e. number of edges for which connected vertices have values with opposite signs in the signal).

III. Using GSP to Analyse Inner Layers of DNNs

Throughout its processing in a DNN, an input vector is transformed in multiple intermediate representations of various dimensions and shapes. In the example of Figure 1, the number of dimensions varies from 10 to a maximum of 270,000. Furthermore, these representations can obey specific constraints (e.g. nonnegativity).

In this paper we propose to use GSP to characterize the evolution of each intermediate representation during the learning process. Using graphs to analyse a dataset is not a novel idea [7], even in the context of GSP [8].

More precisely, we sample M example inputs in each of the C classes, and we monitor their intermediate representations. In the remaining of this work, we shall denote $\mathbf{y}_{c,i}^{e,\ell}$ the intermediate representation of the i-th input of class c at epoch e of the learning process and at the output of layer ℓ. In what follows, we denote a $\mathbf{y}_{c,i}^{e,\ell}$ as $\mathbf{y}$, unless the sub/super indices are needed.

A. Distances and k-Nearest Neighbor Graphs

Consider two vectors $\mathbf{y}, \mathbf{y}' \in \mathbb{R}^d$. We introduce three distances and three associated similarities:

- Euclidean distance and similarity:

$$d_E(\mathbf{y}, \mathbf{y}') \triangleq \sqrt{\sum_{i=1}^{d}(\mathbf{y}_i - \mathbf{y}'_i)^2}\,, s_E = \exp(-d_E)\,.$$

- Normalized Euclidean distance and similarity:

$$d_{NE} \triangleq d_E/\sqrt{d}\,, s_{NE} = \exp(-d_{NE})\,.$$

- Cosine similarity and distance:

$$s_{cos}(\mathbf{y}, \mathbf{y}') \triangleq \frac{\sum_{i=1}^{d}\mathbf{y}_i\mathbf{y}'_i}{\sum_{i=1}^{d}\mathbf{y}_i^2\sum_{i=1}^{d}\mathbf{y}_i'^2}\,, d_{cos} = 1 - s_{cos}.$$

Note that these quantities are extended to tensors by considering their flattened vector counterparts and that the cosine distance and similarity should only be considered for nonnegative inputs.

We call *complete graph* the graph at epoch e and layer ℓ $G_d = \langle V, \mathbf{A}^{e,\ell} \rangle$ where $V = \{(c,i), 1 \leq c \leq C, 1 \leq i \leq M\}$ and $\mathbf{A}_{(c,i)(c',i')}^{e,\ell} = f(\mathbf{y}_{c,i}^{e,\ell}, \mathbf{y}_{c,i}'^{e,\ell})$, f being one of the abovementioned distances or similarities.

We call *k-nearest neighbor graph* associated with $G_d = \langle V, \mathbf{A}^{e,\ell} \rangle$ the graph $G_{d,k} = \langle V, \mathbf{A}^{e,\ell,k} \rangle$ where $\mathbf{A}^{e,\ell,k}$ is obtained from $\mathbf{A}^{e,\ell}$ by keeping only values that are among the k smallest (resp. largest) ones on their row or on their column if f is a distance (resp. similarity).

Considering vertices to be ordered in lexicographic order, we call label signal a binary vector $\mathbf{s}_c \in \mathbb{R}^{CM}$, where all coordinates are zero but the ones between the $((c-1)M+1)$-th coordinate and the cM-th one (i.e. except those belonging to class c).

B. Separation

Separation is a measure of how well separated examples that belong to distinct classes are, in comparison to examples that belong to a same class.

Consider one of the previously introduced graphs. Denote $\mathbf{A}$ its adjacency matrix. Then separation is written:

$$sep^{e,\ell} = \frac{\sum_{c=1}^{C}\sum_{\substack{c'=1\\c'\neq c}}^{C}\mathbf{s}_c^\top supp(\mathbf{A})\mathbf{s}_{c'}}{\sum_{c=1}^{C}\sum_{\substack{c'=1\\c'\neq c}}^{C}\mathbf{s}_c^\top\mathbf{A}\mathbf{s}_{c'}}\frac{\sum_{c=1}^{C}\mathbf{s}_c^\top\mathbf{A}\mathbf{s}_c}{\sum_{c=1}^{C}\mathbf{s}_c^\top supp(\mathbf{A})\mathbf{s}_c}\,,$$

where $supp(\mathbf{A})$ denotes the support of $\mathbf{A}$, that is to say the matrix obtained from $\mathbf{A}$ by replacing each nonzero coordinate with 1. Note that separation is always nonnegative and that it should be lesser than 1 when considering distance graphs, at least for the output layer.

C. Label Smoothness

Label smoothness is a measure of the smoothness of the label signal on a given graph G. Denote by L the Laplacian of G, then:

$$smooth^{e,\ell} = \frac{\sum_{c=1}^{C} \mathbf{s}_c^\top \mathbf{L} \mathbf{s}_{c'}}{M^2 C(C-1)} \ .$$

Note that contrary to separation, a value of label smoothness close to 0 on a similarity graph does not necessarily correspond to the case where all classes are well separated, as the whole similarity graph might be shrinking down.

D. Laplacian Spectrum

The first eigenvalues of the Laplacian of a graph gives an interesting description of how clustered the graph is. In particular an extreme case where the graph is made of several connected components would lead to multiple eigenvalues equal to 0.

Because eigenvalues are linearly impacted by scaling of a graph, we propose to use the normalized Laplacian instead of the combinatorial Laplacian. The normalized Laplacian is obtained from an adjacency matrix $\mathbf{A}$ by computing $\mathcal{L} = \mathbf{I} - \mathbf{D}_\mathbf{A}^{-1/2} \mathbf{A} \mathbf{D}_\mathbf{A}^{-1/2}$. A useful property of the normalized Laplacian is that its spectrum lies between 0 and 2, with at least one eigenvalue that is larger than 1.

Interestingly, this measure does not need the labels of monitored signals, compared to separation and label smoothness. in the case of monitoring intermediate representation of signals in deep neural networks, it would thus be usable for semi-supervised or unsupervised learning problems.

IV. Experiments

We perform experiments using the CIFAR-10 image dataset. It consists of 50,000 training pairs (x_{train}, y_{train}), where the input is of dimensions 32x32x3 (images of 32x32 pixels with 3 primary colors). There are 10 classes. The network performance is then assessed using a testing set of 10,000 images, 1,000 in each class.

We use the architecture depicted in Figure 1 that we train in different conditions:

- A reference condition in which the architecture achieves 85% accuracy on the testing set. In this setting we use data augmentation and dropout with parameter 0.1, resulting in almost no overfitting. Note that this accuracy is not state-of-the-art (see [9] for example) due to the simplicity of the architecture.
- A slightly overfitted condition in which the dropout is removed and no data augmentation is performed during learning phase. Here, accuracy reaches a maximum of 78%.
- An underfitted architecture obtained by dividing by 10 the number of feature maps in each convolutional layer. Accuracy reaches a maximum of 72%.
- Finally an extreme overfitting case where labels of examples have been shuffled arbitrarily. Of course in such conditions accuracy of the testing set remains around 10%, the chance level.

When applicable, we use a 10-nearest neighbor graph.

A. Comparison of Distances on k-Nearest Neighbor Graphs

We perform a first test in which we plot the label smoothness at each layer and each epoch of the training process for the reference condition and for all three similarities in order to compare intermediate representations across layers. In these experiments we use a k-nearest neighbor similarity graph. Results are reported in Figure 2. We observe that Euclidean distance is not well fitted for this experiment, due to the sudden changes in dimensions of intermediate representations that result in orders of magnitude that are hard to compare across layers. A key interest in the cosine distance is that it is upperbounded, but it suffers from the limitation that it only applies to architectures for which intermediate representations are nonnegative. As the normalized Euclidean distance has some useful properties – since it associates layers with comparable values – we use it for upcoming experiments. Note that we ran simulations with separation but results were unusable due to many divisions by 0.

B. Results on Complete Graphs

In Figure 3 we depict separation on the complete distance graph and label smoothness on the complete similarity graph as a function of the number of epochs of training, in all cases for the reference condition. We also depict the evolution of accuracy on both the training and the testing sets. We observe that despite the fact that accuracy is changing, separation appears to be almost constant for all layers except for the last one. Compared with smoothness, we conclude that separation is not informative enough to monitor how examples from distinct classes are detached across the learning process. Also we note that label smoothness on the complete graph seems to be a lot more subject to noise than label smoothness on k-nearest neighbor graphs depicted in Figure 2. For this reason we continue experiments with k-nearest neighbor graphs.

C. Smoothness and Overfitting

A second test we perform consists of comparing label smoothness of k-nearest neighbor graphs under different conditions. Our findings are summarized in Figure 4. There are multiple interesting observations we can draw from these experiments.

1) In all our conditions, we observe a monotonic behavior of smoothness with layer depth. This is intuitive, as we expect first layers to hardly be able to separate training examples from distinct classes, whereas output layers should – as the target of the optimization routine – separate them efficiently. Note that there is a small decrease in the dimension of vectors at each step when going forward in the architecture, and that it could be partially responsible for this phenomenon.

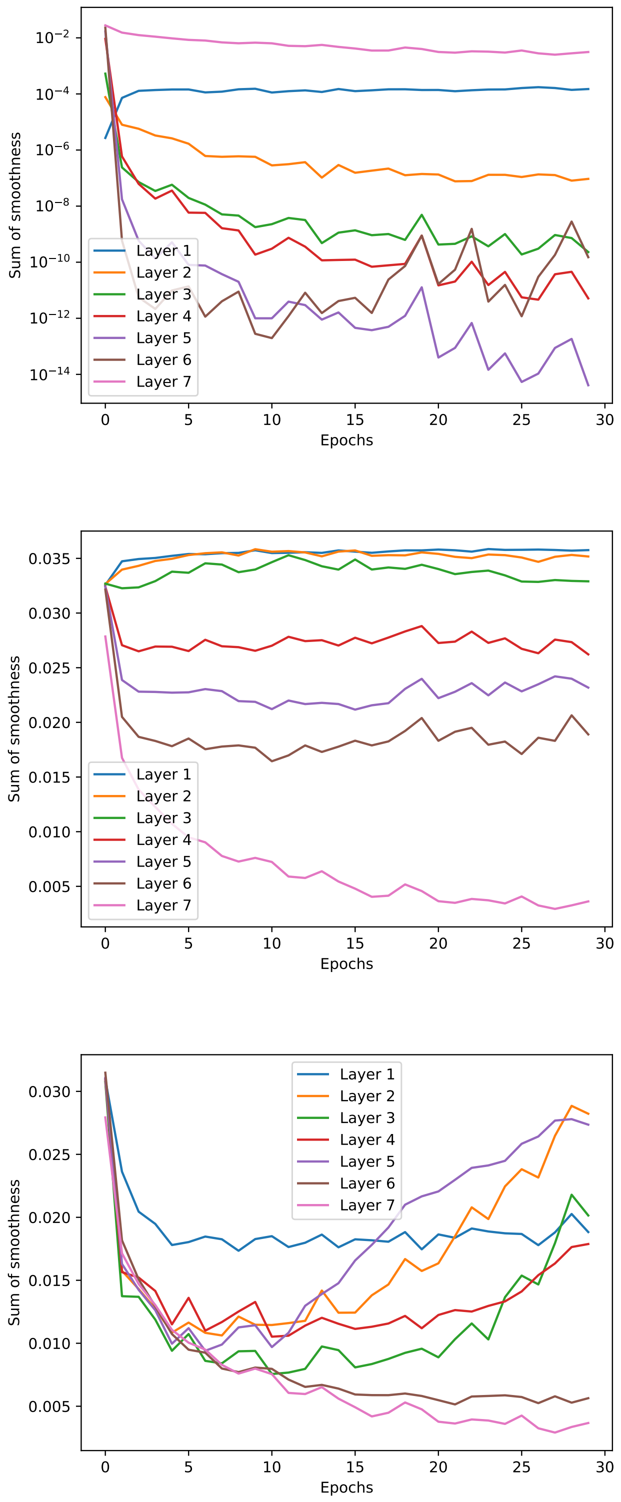


Figure 2. Evolution of label smoothness as a function of the number of epochs of training for the Euclidean distance (first line), the normalized Euclidean distance (second line) and the cosine distance (third line).

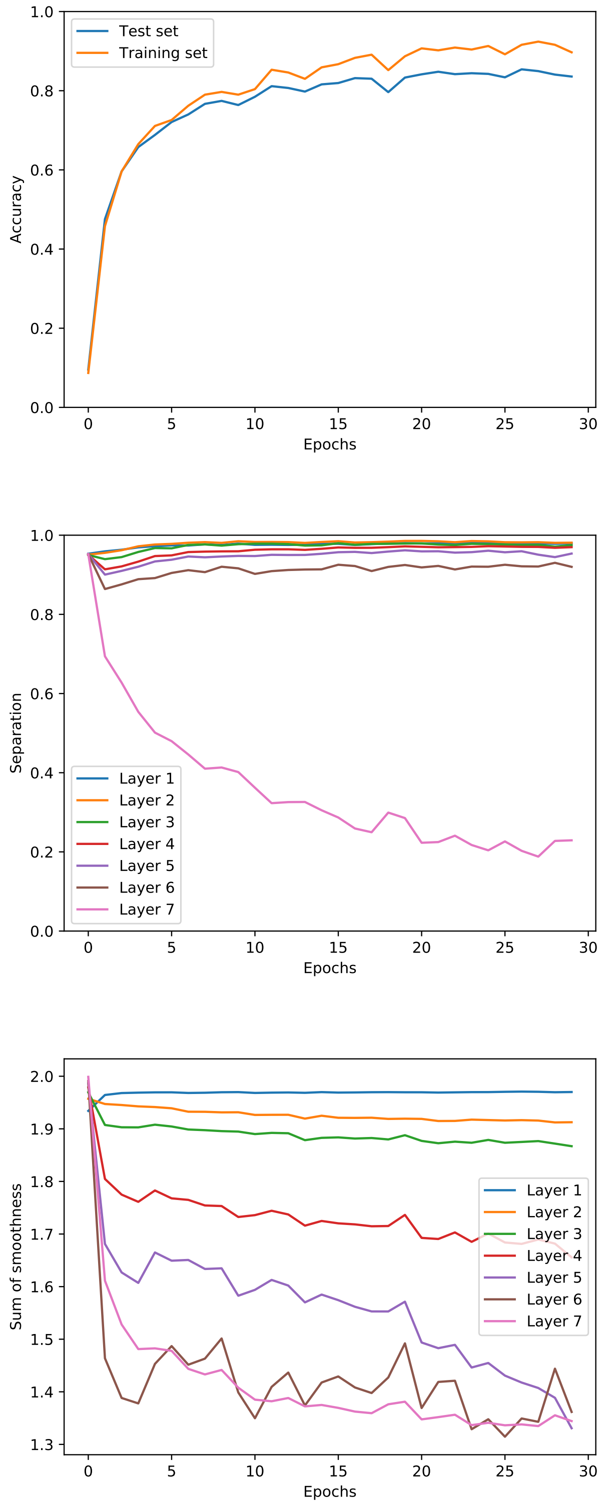


Figure 3. Evolution of accuracy, separation and label smoothness for the normalized Euclidean distance as a function of the number of epochs of training. Note that separation is computed with distance graphs whereas label smoothness is computed with similarity graphs.

Underfitting	Reference	Slight Overfitting	Extreme Overfitting
0.340 ± 0.016	0.827 ± 0.019	0.7960 ± 0.083	0.3231 ± 0.110

Table I
RATIO OF THE DIFFERENCE IN SMOOTHNESS BETWEEN THE OUTPUT LAYER AND THE PENULTIMATE LAYER TO THE LARGEST OF THE TWO. WE PLOT RESULTS FOR DIFFERENT CONDITIONS. RESULTS WERE OBTAINED BY AVERAGING 30 TESTS WITH RANDOM INITIALIZATIONS.

2) In overfitting conditions, multiple layers seem to separate very well training examples, whereas in the reference condition only the output layer is achieving it. This suggests that penultimate layers are less dedicated to the specific training examples, and could explain their great interest for transfer learning [10].

We stress further the latter observation by performing Monte-Carlo experiments. Results are depicted in Table I.

D. Laplacian Spectrum

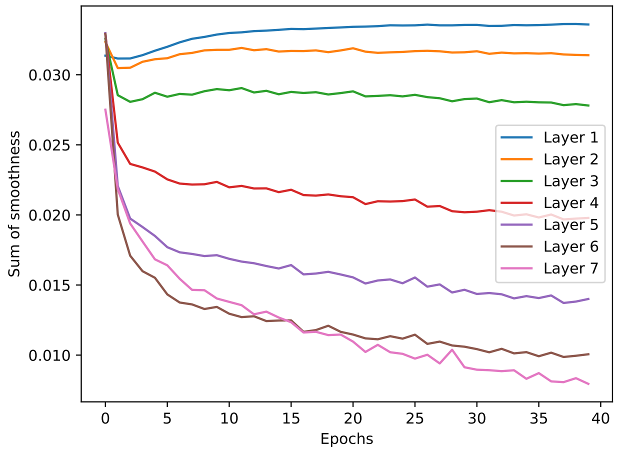
We monitor the first eigenvalues of the normalized Laplacian of the k-nearest neighbor graph in the different conditions. Results are depicted in Figure 5. We observe a relatively smooth evolution of this spectrum for the underfitting and reference conditions. To the contrary, the overfitting conditions lead to sudden changes and only the last layer seems to offer proper clustering of examples. Note that we expect in all cases to see at least the first 10 eigenvalues close to 0 for the output layer, as in all cases accuracy on the training set is close to 100%.

E. Application to Complex Architectures

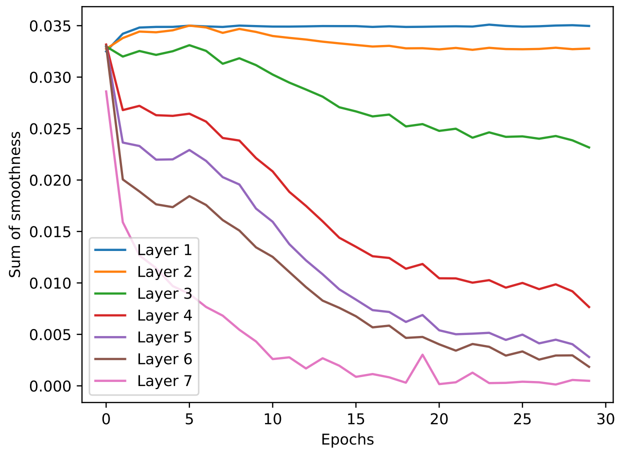
We perform experiments using a more complex architecture named PreActResNet18 [11]. It is made of blocks that are globally arranged linearly as in our model depicted in Figure 1. However each block is made of several paths that process data concurrently, their output being then summed to form the output of the block. This architecture is known to achieve near state-of-the-art performance on CIFAR-10.

Here again, we compare several conditions, and depict the results in Figure 6. In order to interpret these curves, one should first note some important facts.

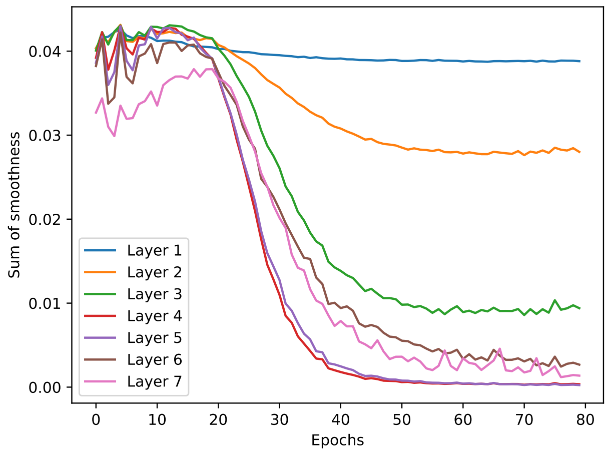
- For some conditions the learning rate is adapted during the learning process, resulting in sudden changes in slopes of the curves.
- We are not necessarily interested in all the intermediate representation at each layer of the architecture, since the parallel computations are thought to produce residuals, which are likely to give chaotic behaviors with respect to smoothness or separation. Rather we focus here on the intermediate representations at the output of blocks along with the initial and final layers, so that each of these representations convey all information that is downstreamingly processed.
- Since some outputs are outputs of blocks and others are outputs of convolutional layers, we talk about "representations" in Figure 6.



a)

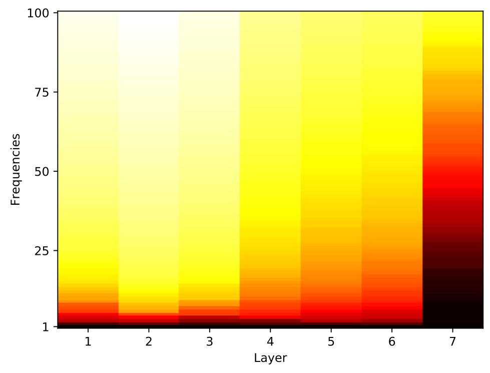


b)

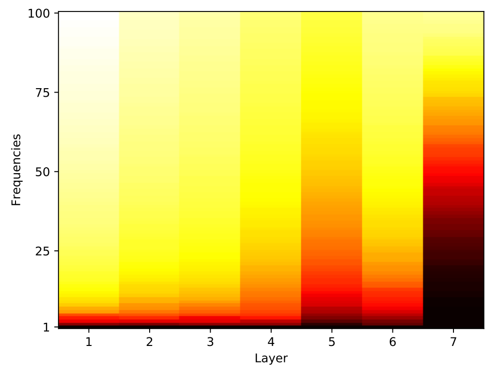


c)

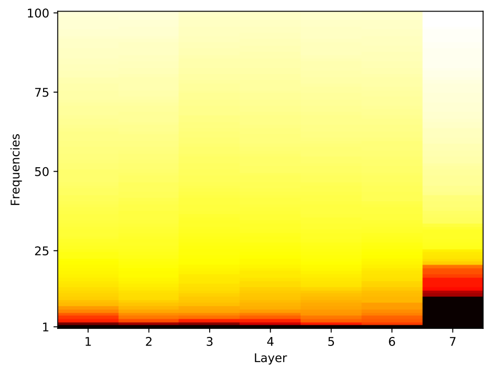
Figure 4. Evolution of label smoothness as a function of the number of epochs for different conditions: a) underfitting, b) slight overfitting and c) extreme overfitting.



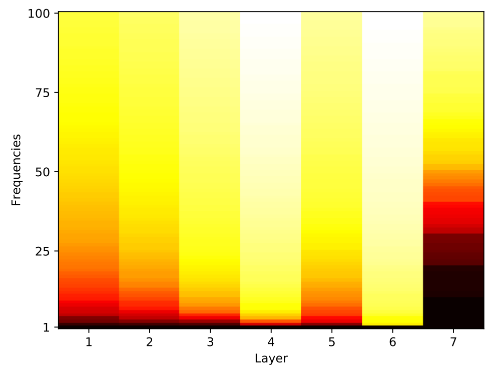
a)



b)



c)



d)

Figure 5. Evolution of first eigenvalues of the nomalized Laplacian across the architecture at the end of the learning process and for different conditions: a) underfitting, b) reference, c) slight overfitting and d) extreme overfitting.

- There are important changes of dimension occuring multiple times throughout the process. It is the case right after representations 3, 5 and 7. As a result, representations should be considered in groups as follows: repr. 1-2-3, repr. 4-5, repr. 6-7, and repr. 8-9-10.

Interestingly, we observe significant changes between the different architectures, in particular in the group 8-9-10. In the reference case the three layers have reasonably similar label smoothness, whereas in other conditions we see important gaps, in particular between representations 8 and 9.

Similarly to the simple architecture, the lack of a gap between label smoothness between the penultimate representation and the output one seems to be a good indicator of overfitting. Indeed, the reference condition is arguably slightly overfitting, considering the very high score on the training set.

Another interesting observation we can draw from these experiments is the impact of the learning rate on the slopes of the curves. This is particularly true for the reference condition where the impact is considerable. As a matter of fact it seems that the sudden change in the learning rate specializes the last group in clustering properly the examples whereas it effects the contrary for previous representations.

Another interesting observation is the fact the label smoothness continues to evolve by a great extent even if training and test accuracies remain constant. This motivate the idea of using label smoothness to properly choose when to adapt learning rate during training.

V. Conclusion

In this paper we introduced a way to monitor intermediate representations of deep neural networks using graph signal processing. Our initial findings suggest that smoothness of the label signals on a k-nearest neighbor graph obtained using normalized Euclidean distance is a good measure of separation of classes in these intermediate representations.

Future work include performing tests on state-of-the-art architectures, checking reproducibility with other datasets and designing optimization costs that specifically aim at decreasing smoothness on specific layers.

References

[1] S. Mallat, “Understanding deep convolutional networks,” *Phil. Trans. R. Soc. A*, vol. 374, no. 2065, p. 20150203, 2016.

[2] J. Viswanathan, F. Rémy, N. Bacon-Macé, and S. J. Thorpe, “Long term memory for noise: evidence of robust encoding of very short temporal acoustic patterns,” *Frontiers in neuroscience*, vol. 10, p. 490, 2016.

[3] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[5] S. M. Moosavi Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, no. EPFL-CONF-218057, 2016.

[6] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, “Graph signal processing,” *arXiv preprint arXiv:1712.00468*, 2017.

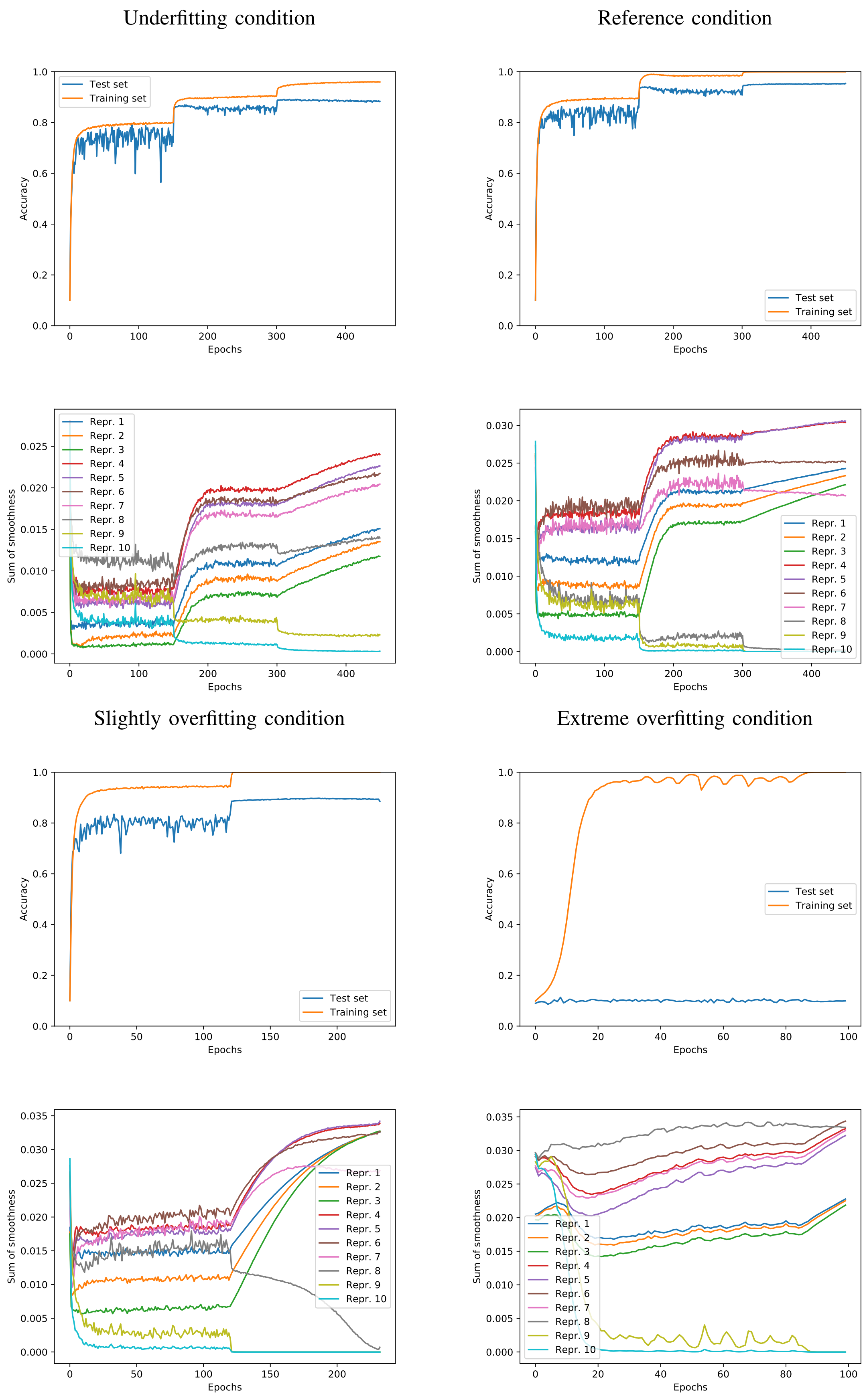


Figure 6. Comparison of accuracy and label smoothness for PreActResNet18 under different conditions.

[7] X. Zhu, J. Lafferty, and R. Rosenfeld, “Semi-supervised learning with graphs,” Ph.D. dissertation, Carnegie Mellon University, language technologies institute, school of computer science, 2005.

[8] A. Gadde, A. Anis, and A. Ortega, “Active semi-supervised learning using sampling theory for graph signals,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 492–501.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] Y. Bengio, “Deep learning of representations for unsupervised and transfer learning,” in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, 2012, pp. 17–36.

[11] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.