Building Graph Representations of Deep Vector Embeddings

Dario Garcia-Gasulla
Barcelona Supercomputing Center (BSC)
dario.garcia@bsc.es

Armand Vilalta
Barcelona Supercomputing Center (BSC)
armand.vilalta@bsc.es

Ferran Parés
Barcelona Supercomputing Center (BSC)
ferran.pares@bsc.es

Jonatan Moreno
Barcelona Supercomputing Center (BSC)
jonatan.moreno@bsc.es

Eduard Ayguadé
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya
eduard.ayguade@bsc.es

Jesús Labarta
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya
jesus.labarta@bsc.es

Ulises Cortés
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya
ia@cs.upc.edu

Toyotaro Suzumura
Barcelona Supercomputing Center (BSC)
IBM T.J. Watson
tsuzumura@us.ibm.com

Abstract

Patterns stored within pre-trained deep neural networks compose large and powerful descriptive languages that can be used for many different purposes. Typically, deep network representations are implemented within vector embedding spaces, which enables the use of traditional machine learning algorithms on top of them. In this short paper we propose the construction of a graph embedding space instead, introducing a methodology to transform the knowledge coded within a deep convolutional network into a topological space (*i.e.*, a network). We outline how such graph can hold data instances, data features, relations between instances and features, and relations among features. Finally, we introduce some preliminary experiments to illustrate how the resultant graph embedding space can be exploited through graph analytics algorithms.

1 Introduction

Deep learning models build large and rich data representations by finding complex patterns within large and high-dimensional datasets. At the end of a deep learning training procedure, the learnt model can be understood as a data representation language, where the pattern learnt by each neuron within the deep model represents a word of such language. Extracting and reusing the patterns learnt by a deep neural network (DNN) is a subfield of deep learning known as transfer learning. Transfer learning from a pre-trained DNN can be used to initialize the training of a second DNN from a non-random state, improving performance over randomly initialized networks [22, 2, 10], and also enabling the training of DNNs for domains with limited amount of data [8, 20]. These two settings, where the purpose of the transfer learning process is to train a second DNN, are cases of transfer learning for fine tuning. A different purpose of transfer learning is to extract deep representations so that alternative machine learning methods can be run on top of those. This is commonly known as *transfer learning for feature extraction*, and is the main topic of this paper.

Extracted DNN representations are typically implemented through vectors, where the length of the vector equals to the number of neural features being used. These vector embedding spaces have been used to feed classifiers based on the instance-attribute paradigm (*e.g.*, Support Vector Machines) [1, 7]. Instead, in this paper we propose a *graph* based representation of those same embeddings spaces, with

the goal of running a different family of algorithms; those based on the instance-instance paradigm, such as community detection algorithms. Graph or network based algorithms focus on the associations among instances to find topologically coherent patterns. These are significantly different from the patterns that can be found using algorithms focused on the associations among instances and attributes. This paper describes a methodology for building a graph representation of neural network embeddings (in §2), and reports the performance of a community detection algorithm processing the resultant graph (in §3).

2 Graph Representation of Vector Embeddings

Vector embeddings are often built by capturing the output of a single layer close to the output of a deep convolutional neural network (CNN), typically a fully-connected layer [18, 9, 5, 11, 16]. However, earlier layers can also be useful for characterizing data, particularly when this data is not strongly related with the data that was used for pre-training the model [1]. Another difference between later layers and the rest in a typical CNN trained for classification is that layers close to the output exhibit a strongly discriminative behavior, activating sporadically and in a crisp manner (either strongly activating or not at all). In contrast, neurons of layers further from the output show a more descriptive behavior, activating frequently and in a fuzzy way [6].

In this paper we generate a graph representation which captures data properties in a topological space (*i.e.*, through vertices and their associations). Using a single layer embedding for that purpose would result in a rather poor topology, as fully-connected layers represent but a portion of all patterns learnt by the DNN, and only a small subset of neurons activate for each data instance. To guarantee that the graph representation contains a topology rich enough as to empower network analysis algorithms we use the full-network embedding [7], which produces a vector embedding including all convolutional and fully-connected layers of a CNN. This results in a much larger embedding space (composed by tens of thousands of dimensions), allowing us to generate larger and richer graphs.

Next we briefly introduce the full-network embedding, and at the end of this section we describe how we generate a graph to capture the embedding space within its topology.

2.1 Full-network Embedding

The full-network embedding (FNE) generates a representation of an input data instance by capturing the activations it produces at every convolutional and fully-connected layer within a CNN. In order to integrate the features found at layers of different depth, size and nature, the FNE includes a set of processing steps. An overview of these can be seen in Figure 1.

After the extraction of activations, the FNE applies a spatial average pooling on the activations coming from convolutional layers. As a result, neurons of convolutional layers will generate a single value in the embedding (as neurons from fully-connected layers do). After the spatial pooling, the FNE includes a feature standardization step. The goal of this transformation is to normalize the values of the different neurons, so that each neuron has a coherent range of activations regardless of its type or location within the CNN. Otherwise, the activations from the layers close to the output would dominate the representation, as neurons from these layers typically activate with much more strength.

Finally, the FNE discretizes features, mapping all values to either -1, 0 or 1. This process reduces noise and regularizes the embedding space. In the FNE, this discretization is done with a pair of constant thresholds $(-0.25, 0.15)$ which determine if a feature is relevant by presence (1 implies an abnormally high activation) or is relevant by absence (-1 implies an abnormally low activation) for a given input data instance [7]. In our experiments we set more demanding thresholds $(-2.0, 2.0)$, to make sure that the degree of sparsity of the graph is appropriate for network analysis methods.

2.2 Graph Representation

The FNE generates a vector representation of each data input, where feature values are either -1, 0 or 1. Those values represent the relevance of a specific neural filter for a given data input, indicating feature
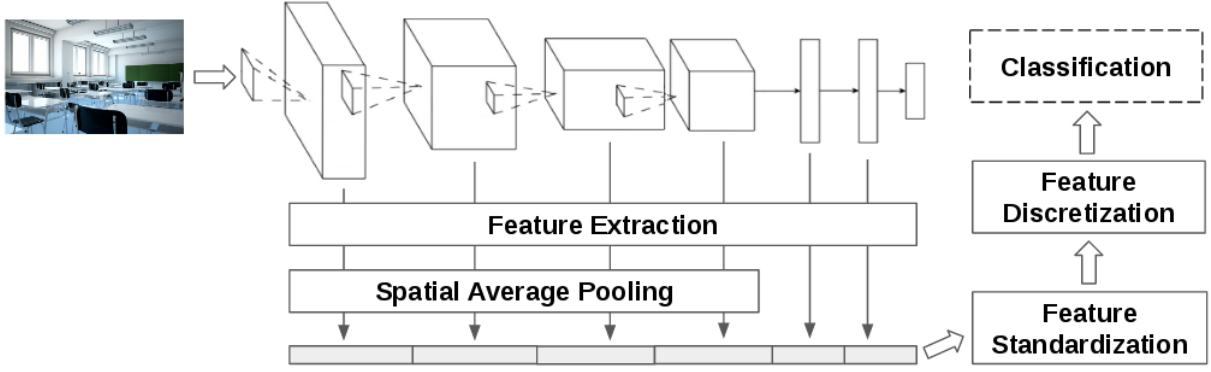
Figure 1: Overview of the Full-network embedding generation workflow.

relevance by absence, feature irrelevance, and feature relevance by presence, respectively. In this paper we consider building a topology based representation (*i.e.*, a graph) of such embedding, and processing it through algorithms exploiting instance-instance relations (*e.g.*, community detection) to exploit the information encoded topologically.

2.2.1 Vertices

Let us start defining what composes the vertices (V) of our graph representation G. From the discrete three-valued full-network embedding we extract both data instances (*e.g.*, images of a dataset) and model features (*i.e.*, neural filters of the CNN). Each data instance is represented in the graph G as a unique vertex (of type image vertex or V_i). Features however, may be relevant by presence or by absence. We choose to capture this dichotomy by creating two vertices in the graph for each feature; one of those vertices will represent the relevant activation of the feature (positive feature vertex or V_f^+), while the other will represent the relevant lack of activation of the feature (negative feature vertex or V_f^-). As a result, the number of image vertices in the graph ($|V_i|$) will be equal to the number of input data images, while the number of positive and negative feature vertices ($|V_f^+|$ and $|V_f^-|$ separately) will be equal to the number of features in the original vector embedding.

2.2.2 Image-Feature Edges

Let us now define the edges (E) of our graph representation G. The implementation of edges between image vertices and feature vertices (E_{if}) is straight-forward. We add an edge between a vertex image v_i and a positive feature vertex v_f^+ when the corresponding value in the full-network embedding is 1. Analogously, we add an edge between a vertex image v_i and a negative feature vertex v_f^- when the corresponding value in the full-network embedding is -1. Edges between image vertices and positive feature vertices indicate that the occurrence of the feature is relevant for describing the image, while edges between image vertices and negative feature vertices indicate that the no occurrence of the feature is relevant for describing the image. Feature values of 0 identify irrelevancy, which is why these are not coded into the graph.

2.2.3 Feature-Feature Edges

One of the benefits of using a graph-based representation for the embedding is that, unlike vector representations, it allows us to extend the embedding space to include feature-feature relations. In the graph this corresponds to the creation of edges between feature vertices (E_{ff}). To identify relations between features we consider the weights ($\mathcal{W}$) of the pre-trained CNN. If the weight associating a neuron f_{l+1} with a neuron from the previous layer f_l ($\mathcal{W}(f_{l+1}, f_l)$) is abnormally high in the context of all weights of f_{l+1} with the previous layer ($\mathcal{W}(f_{l+1}, F_l)$), there exists a positive correlation between both neurons.

Similarly, if $\mathcal{W}(f_{l+1}, f_l)$ is abnormally low in the context of $\mathcal{W}(f_{l+1}, F_l)$, there exists a negative correlation between neurons f_{l+1} and f_l. To identify what counts as abnormally high or abnormally low, we compute the mean (μ) and the standard deviation (σ) of the weights that associate a feature f_{l+1} from a given layer with every feature from the previous layer $f_i \in F_l$ (in the case of convolutional filters we previously sum all the weights of the receptive field). If the weight associating f_{l+1} with one of the features of the previous layer is above that mean a given number of standard deviations, then such relation is abnormally high (see Equation 1). An abnormally low relation is defined analogously.

We implement positive correlations in the graph by adding an edge between the positive feature vertex of f_l and the positive feature vertex of f_{l+1} ($e_{ff}(v_{f_l}^+, v_{f_{l+1}}^+)$), and another edge between the negative feature vertex of f_l and the negative feature vertex of f_{l+1} ($e_{ff}(v_{f_l}^-, v_{f_{l+1}}^-)$). Negative correlations are implemented by adding an edge between the positive feature vertex of f_l and the negative feature vertex of f_{l+1} ($e_{ff}(v_{f_l}^+, v_{f_{l+1}}^-)$), and another edge between the negative feature vertex of f_l and the positive feature vertex of f_{l+1} ($e_{ff}(v_{f_l}^-, v_{f_{l+1}}^+)$).

$$
\begin{aligned}
\{e_{ff}(v_{f_l}^+, v_{f_{l+1}}^+), e_{ff}(v_{f_l}^-, v_{f_{l+1}}^-)\} \in E_{ff} &\quad \text{iff} \quad \mathcal{W}(f_{l+1}, f_l) > \mu(\mathcal{W}(f_{l+1}, F_l)) + \sigma(\mathcal{W}(f_{l+1}, F_l)) * k \\
\{e_{ff}(v_{f_l}^+, v_{f_{l+1}}^-), e_{ff}(v_{f_l}^-, v_{f_{l+1}}^+)\} \in E_{ff} &\quad \text{iff} \quad \mathcal{W}(f_{l+1}, f_l) < \mu(\mathcal{W}(f_{l+1}, F_l)) - \sigma(\mathcal{W}(f_{l+1}, F_l)) * k
\end{aligned}
\tag{1}
$$

The parameter k of Equation 1 regulates the sparsity of feature-feature edges. A higher k value will only accept edges between very strongly connected pairs of neurons, as the associated weights must be a larger number of standard deviations above the mean. In all our experiments we set k to 1.5.

At this point we can finally define our graph representation as $G = (V, E)$ where $E = E_{if} \cup E_{ff}$ and $V = V_i \cup V_f^+ \cup V_f^-$.

3 Experiments

To evaluate our graph representation of the deep embedding space we use the VGG16 CNN architecture [21], pre-trained on the ImageNet [17] dataset. We process four different datasets through this pre-trained model. Details on the datasets are shown in Table 1:

- The MIT Indoor Scene Recognition dataset [15] (*mit67*) consists of different indoor scenes to be classified in 67 categories.

- The Oxford Flower dataset [12] (*flowers102*) is a fine-grained dataset consisting of 102 flower categories.

- The Describable Textures Dataset [4] (*textures*) is a database of textures categorized according to a list of 47 terms inspired from human perception.

- The Oulu Knots dataset [19] (*wood*) contains knot images from spruce wood, classified according to Nordic Standards.

For each of the images of those datasets we obtain the full-network embedding. In the case of the VGG16 architecture, the embedding generates vectors of 12,416 features. Based on those, we build the graph representation, as previously described.

We explore the graph-representation by running a community detection algorithm on top of it. Particularly, we use the Fluid Communities (FluidC) algorithm [13]. We chose this algorithm because its based on the efficient label propagation methodology while outperforming the traditional LPA algorithm, because it allows us to specify the number of clusters we wish to find, and because it can be easily adapted to the specific needs of our experiments. These specific needs regard mostly cluster initialization and maintainance. Since the graph is composed by both images and features vertices, but only images have an associated label, clusters should be evaluated considering only the image vertices found in a final community. For this same reason, we must ensure that all communities found contain at least one image

Table 1: Properties of all datasets computed

Dataset	#Images	#Classes	#Images per class
mit67	6,700	67	100
flowers102	8,189	102	40 - 258
textures	5,640	47	120
wood	438	7	14 - 179

Table 2: Properties of the graphs built from the deep embedding spaces, and quality of the communities found by the FluidC algorithm measured in NMI and AMI.

Dataset	*mit67*	*flowers102*	*textures*	*wood*
$|V|$	30,186	33,010	30,459	25,259
$|E|$	8,396,939	9,654,464	8,561,314	5,610,832
NMI	0.44	0.54	0.42	0.26
AMI	0.36	0.46	0.37	0.20

vertex. This is done by modifying the FluidC algorithm, forcing it to initialize communities on image vertices, and by making sure that a community contains at least one image vertex at all times.

We measure the quality of the found clusters by measuring the similarity between the found communities and the original dataset labels. We use both the normalized mutual information measure (NMI) and the adjusted mutual information (AMI). The properties of the graph generated for each dataset and the performance results are shown in Table 2.

All experiments were done using the VGG16 model pre-trained on ImageNet2012, freely available online. The feature extraction process was done with Caffe. The execution of the graph algorithm was done using NetworkX v2.0, which includes FluidC.

4 Related Work

The relation between graphs and deep neural networks have been previously explored, but most contributions do so from a different perspective. While our proposal is to obtain a graph representation of the embedding produced by a CNN when processing an image, most related work focuses on training DNNs for processing graph data. For example, DeepWalk [14] uses random walks in a graph (*e.g.*, Flickr or YouTube networks) to feed a SkipGram model, and then evaluate community detection methods on those graphs. Similarly to DeepWalk, the work of Cao et al. [3] also processes graphs as input, but it uses a probabilistic approach on weighted graphs to feed an autoencoder. In contrast, we are performing community detection on a dataset of images, something that, to the best of our knowledge, had not been done before.

5 Conclusions

The presented methodology is a first step towards building graph-based representations of deep CNN embeddings. We detail how to include in such a graph both images and features, and how to topologically codify image-feature and feature-feature relations. By doing so we make deep knowledge available to a large set of learning algorithms (network analysis tools) which may exploit those representations in a completely different manner.

The results reported are encouraging, as a topology based algorithm such as FluidC is capable of identifying relevant communities of images using only topological information. The clusters that can

be found through network analysis tools are significantly different than the clusters that can be found through more traditional algorithms (*e.g.*, Kmeans) running on a vector representation. Indeed, while algorithms like FluidC exploit the paths among vertices, algorithms like Kmeans are based on distance measures which can hardly integrate the information of all possible paths in a graph. The possibility of running a different type of algorithms makes this novel approach interesting, as it opens the door at exploiting and reusing the knowledge coded within deep pre-trained neural models in a completely new way. This is but a small step towards a better understanding of deep representations, and how to exploit all the knowledge these encode for a wider variety of purposes.

6 Future Work

The experiments shown in this paper are highly exploratory, with the purpose of validating the feasibility of the hypothesis (*i.e.*, that deep neural representations can be implemented as a graph coherently). Analytically, it would be desirable to compare the performance of the proposed approach with alternatives, such as applying a clustering algorithm on the original vector embeddings. However, clustering evaluation is a controversial topic, as there is not a single solution which may be considered to be the universal ground truth.

A more relevant evaluation of the approach should be based on an analysis of the semantics captured by the graph. For that purpose we are considering the extension of the model to include directed edges, weighted edges, and ontological relations. By doing so one could execute inference methods on top of the representation, and measure how rich and usable the semantics captured in the graph actually are.

Acknowledgements

References

[1] Azizpour, H., A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson (2016). Factors of transferability for a generic convnet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 38*(9), 1790–1802.

[2] Branson, S., G. Van Horn, S. Belongie, and P. Perona (2014). Bird species categorization using pose normalized deep convolutional nets. *arXiv preprint arXiv:1406.2952*.

[3] Cao, S., W. Lu, and Q. Xu (2016). Deep neural networks for learning graph representations. In *AAAI*, pp. 1145–1152.

[4] Cimpoi, M., S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi (2014). Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3606–3613.

[5] Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *Icml*, Volume 32, pp. 647–655.

[6] Garcia-Gasulla, D., F. Parés, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura (2017). On the behavior of convolutional nets for feature extraction. *arXiv preprint arXiv:1703.01127*.

[7] Garcia-Gasulla, D., A. Vilalta, F. Parés, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzu-mura (2017). An out-of-the-box full-network embedding for convolutional neural networks. *arXiv preprint arXiv:1705.07706*.

[8] Ge, W. and Y. Yu (2017). Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. *arXiv preprint arXiv:1702.08690*.

[9] Gong, Y., L. Wang, R. Guo, and S. Lazebnik (2014). Multi-scale orderless pooling of deep convolu-tional activation features. In *European conference on computer vision*, pp. 392–407. Springer.

[10] Liu, C., Y. Cao, Y. Luo, G. Chen, V. Vokkarane, and Y. Ma (2016). Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment. In *International Conference on Smart Homes and Health Telematics*, pp. 37–48. Springer.

[11] Mousavian, A. and J. Kosecka (2015). Deep convolutional features for image based retrieval and scene categorization. *arXiv preprint arXiv:1509.06033*.

[12] Nilsback, M.-E. and A. Zisserman (2008). Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Confer-ence on*, pp. 722–729. IEEE.

[13] Parés, F., D. Garcia-Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura (2017). Fluid communities: A community detection algorithm. *arXiv preprint arXiv:1703.09307*.

[14] Perozzi, B., R. Al-Rfou, and S. Skiena (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM.

[15] Quattoni, A. and A. Torralba (2009). Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 413–420. IEEE.

[16] Ren, R., T. Hung, and K. C. Tan (2017). A generic deep-learning-based approach for automated surface inspection. *IEEE Transactions on Cybernetics*.

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

[18] Sharif Razavian, A., H. Azizpour, J. Sullivan, and S. Carlsson (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813.

[19] Silvén, O., M. Niskanen, and H. Kauppinen (2003). Wood inspection with non-supervised cluster-ing. *Machine Vision and Applications 13*(5), 275–285.

[20] Simon, M. and E. Rodner (2015). Neural activation constellations: Unsupervised part model dis-covery with convolutional networks. In *Proceedings of the IEEE International Conference on Com-puter Vision*, pp. 1143–1151.

[21] Simonyan, K. and A. Zisserman (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[22] Xu, Z., S. Huang, Y. Zhang, and D. Tao (2015). Augmenting strong supervision using web data for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2524–2532.