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 [1, 2, 3], and also enabling the training of DNNs for domains with limited amount of data [4, 5]. 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) [6, 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 fully-connected layer close to the output of the neural network. Building a graph representation from such an embedding would result in a rather poor topology, as fully-connected layers represent but a portion of all patterns learnt by the DNN. 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 deep convolutional neural network (CNN). This results in a much larger embedding space (composed by tens of thousands of dimensions), allowing us to generate larger and richer graphs.

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. 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 or absence 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 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 encoded information.
2.2.1 Vertices

Let us start defining what composes the vertices of our graph representation. 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 $|V_f^-|$) 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 straightforward. 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 among feature vertices ($E_{ff}$). To identify relations between features we consider the weights ($W$) of the pre-trained CNN. If the weight associating a neuron $f_{l+1}$ with a feature $f_l$ from the previous layer ($W(f_{l+1}, f_l)$) is abnormally high in the context of all weights of $f_{l+1}$ with the previous layer ($W(f_{l+1}, F_l)$), there exists a positive correlation between both neurons. Similarly, if $W(f_{l+1}, f_l)$ is abnormally low in the context of $W(f_{l+1}, F_l)$, there exists a negative correlation between both neurons. To identify what counts as abnormally high or abnormally low, we compute the mean ($\mu$) and the standard deviation ($\sigma$) of the weights that associate a feature $f_{l+1}$ from a given layer with every feature from the previous layer $f_l \in F_l$ (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 $f_{l+1}$, and another edge between the negative feature vertex of $f_l$ and the negative feature vertex of $f_{l+1}$. Negative correlations are implemented by adding an edge between the positive feature vertex of $f_l$ and the negative feature vertex of $f_{l+1}$, and another edge between the negative feature vertex of $f_l$ and the positive feature vertex of $f_{l+1}$.

\[
\begin{align*}
\{e_{ff}(v_f^+, v_{f_{l+1}}^+), e_{ff}(v_f^-, v_{f_{l+1}}^-)\} & \in E_{ff} \quad \text{iff} \quad W(f_{l+1}, f_l) > \mu(W(f_{l+1}, F_l)) + \sigma(W(f_{l+1}, F_l)) \ast k \\
\{e_{ff}(v_f^+, v_{f_{l+1}}^+), e_{ff}(v_f^-, v_{f_{l+1}}^-)\} & \in E_{ff} \quad \text{iff} \quad W(f_{l+1}, f_l) < \mu(W(f_{l+1}, F_l)) - \sigma(W(f_{l+1}, F_l)) \ast k
\end{align*}
\]

(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
Table 1: Properties of the graphs built from the deep embedding spaces, and quality of the communities found by the FluidC algorithm measured in NMI.

| 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 |

A larger number of standard deviations above the mean. In all our experiments we set $k$ to 1.5. Finally, we define our graph representation $G = (V, E)$ 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 [8], pre-trained on the ImageNet [9] dataset. We process four different datasets through this pre-trained model (mit67 [10], flowers102 [11], textures [12] and wood [13]) and obtain the full-network embedding for each of those. 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 [14]. We evaluate the algorithm performance by measuring the similarity between the found communities and the original dataset labels, using the normalized mutual information measure (NMI). Since the graph is composed by both images and features vertices, but only images have an associated label, the NMI is measured considering only the image vertices found in a community. For this same reason, we must ensure that all communities found contain at least one image 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. The properties of the graph generated for each dataset and the performance results are shown in Table 1.

4 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. This alone 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.
References

[1] Zhe Xu, Shaoli Huang, Ya Zhang, and Dacheng Tao. Augmenting strong supervision using web data for fine-grained categorization. In Proceedings of the IEEE International Conference on Computer Vision, pages 2524–2532, 2015.

[2] Steve Branson, Grant Van Horn, Serge Belongie, and Pietro Perona. Bird species categorization using pose normalized deep convolutional nets. arXiv preprint arXiv:1406.2952, 2014.

[3] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, and Yunsheng Ma. Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment. In International Conference on Smart Homes and Health Telematics, pages 37–48. Springer, 2016.

[4] Weifeng Ge and Yizhou Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. arXiv preprint arXiv:1702.08690, 2017.

[5] Marcel Simon and Erik Rodner. Neural activation constellations: Unsupervised part model discovery with convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision, pages 1143–1151, 2015.

[6] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. Factors of transferability for a generic convnet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(9):1790–1802, 2016.

[7] Dario Garcia-Gasulla, Armand Vilalta, Ferran Parés, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. An out-of-the-box full-network embedding for convolutional neural networks. arXiv preprint arXiv:1705.07706, 2017.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3):211–252, 2015.

[10] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 413–420. IEEE, 2009.

[11] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In Computer Vision, Graphics & Image Processing, 2008. ICVGIP’08. Sixth Indian Conference on, pages 722–729. IEEE, 2008.

[12] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3606–3613, 2014.

[13] Olli Silvén, Matti Niskanen, and Hannu Kauppinen. Wood inspection with non-supervised clustering. Machine Vision and Applications, 13(5):275–285, 2003.

[14] Ferran Parés, Dario Garcia-Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. Fluid communities: A community detection algorithm. arXiv preprint arXiv:1703.09307, 2017.