Hybrid AI using Graph Neural Networks in Scene Classification

Maaike H.T. de Boer[a], Fieke Hillerström[b]

[a] TNO Data Science, Anna van Buerenplein 1, 2595 DA, The Hague, The Netherlands
[b] TNO Intelligent Imaging, Oude Waalsdorperweg 63, 2597 AK, The Hague, The Netherlands

Abstract

In Hybrid AI, machine learning and knowledge engineering are combined to have the best of both worlds. Insights obtained from data are combined with complementary expert knowledge, which can be represented in a graph structure. Graph networks are a recently new development in machine learning and cover methods that learn on graph structured data. This paper researches how knowledge can be incorporated in graph networks for the use case of scene classification. The aim is to detect novel scenes, of which only a few examples and noisy object detections are available. The results show that both using a graph network and adding knowledge can improve performance, however, this is not always necessarily the case. The novelty of this paper is threefold: 1. Using GNNs for scene classification; 2. Combining data and knowledge in GNNs by constructing one input graph; 3. Using GNNs in cases with few training samples and noisy inputs.

Keywords

Hybrid AI, Graph Neural Networks, Scene Classification

1. Introduction

Past AI approaches have not been very successful in open world situations [1]. Systems fail due to a high sensitivity to deviations from assumptions about the external world. Recurrent updating of the world models is necessary, as the world changes. Case-based reasoning is employed to adapt previous solutions to current situations, however in practice this approach becomes a retrieval of similar past cases without adaptation. Deep learning based methods for situational awareness are characterized by a dependence on massive amounts of training data and often lack transferability to unseen situations.

An approach to overcome these problems is the use of Hybrid AI, which combines data driven approaches with expert knowledge. The knowledge is used to fill the information gaps that are present in the data and can be updated in an iterative manner. Expert knowledge can be represented in a knowledge graph, by representing entities as nodes and their relations as edges. The knowledge is combined with actual data (observations of the entities) in a graph structure. We combine this graph structured knowledge with Graph Neural Networks (GNNs),

In A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, F. van Harmelen (Eds.), Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021) - Stanford University, Palo Alto, California, USA, March 22-24, 2021.

maaike.deboer@tno.nl (M.H.T.d. Boer); fieke.hillerstrom@tno.nl (F. Hillerström)
0000-0002-2775-8351 (M.H.T.d. Boer); 0000-0003-1301-3073 (F. Hillerström)

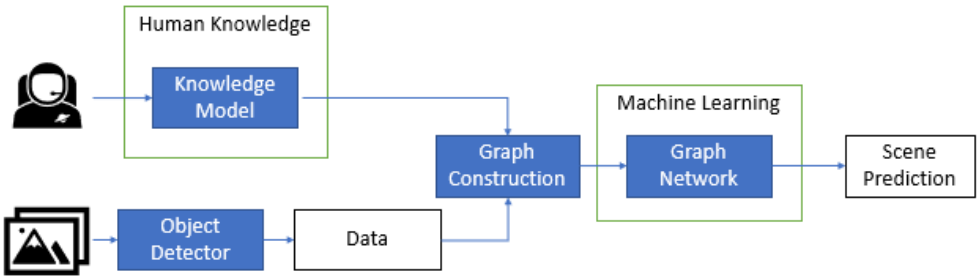

Figure 1: Our Hybrid AI approach in which the image data is combined with expert knowledge.

a machine learning technique that operates on graphs. This new Deep Learning technique showed a lot of potential and the link with graph-structured knowledge enables possibilities for explainability. GNNs are able to handle new nodes, which makes them suitable for dynamic knowledge graphs.

Our research focuses on scene classification in an open world setting; predict the type of room based on an image, with minimal human involvement. The number of training images is low, to mimic the open world setting where for new scenes only a few training images are available. Expert knowledge and scene images are the input of our hybrid approach, which is shown in Figure 1. The knowledge determines which objects are expected in a scene and for these objects detectors are trained on the fly, using images scraped from the internet. Because of this automatic process, the resulting object detections will be noisy. In our research, we mimic these noisy object detections by taking the predictions of a Faster-RCNN finetuned on the MSCOCO dataset [2] with a confidence threshold of 0.1. The object detections are combined with the human knowledge into one graph. In this graph, all detected objects in an image are represented as nodes (attributed with the number of detections) and all-to-all edges are used to link the nodes. Knowledge is added as a separate scene node for which an edge between the object nodes which are evidence for that scene is drawn. The graph is the input of the GNN, the machine learning part of the system. The GNN makes a prediction about the scene type, and is tested on the ADE Scene dataset [3].

The novelty of this paper is threefold: 1. Using GNNs for scene classification; 2. Combining data and knowledge in GNNs by constructing one input graph; 3. Using GNNs in cases with few training samples and noisy inputs.

This paper is organised as follows. The next section describes the related work, mainly focused on graph neural networks. Section 3 contains the experimental setup, with explanation of the dataset, how knowledge is created, which methods are compared, and the evaluation metrics. Section 4 presents the results, section 5 follows with the discussion and the paper is concluded in section 6.

2. Related Work

Current state of the art in deep learning methods use (Convolutional) Neural Networks ((C)NN) [4, 5], where the multi-layer filters used for feature extraction are automatically learned using

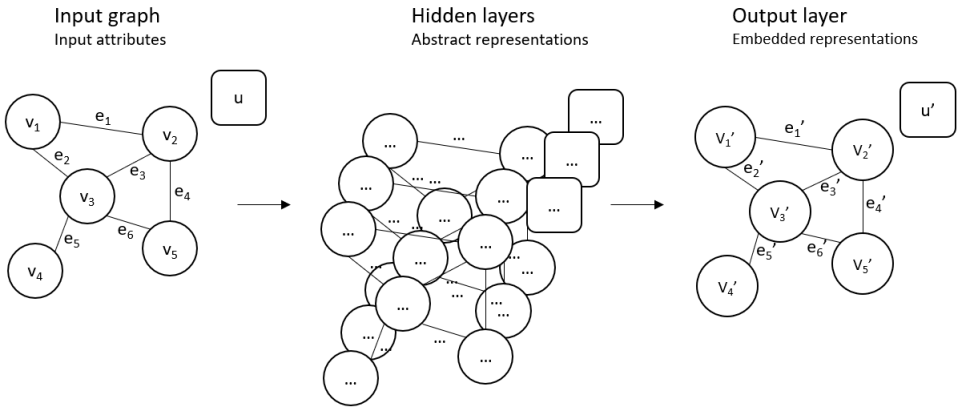

Figure 2: Representation of the GNN processing structure. An embedded representation of the graph is calculated (indicated by the 's), by applying neural networks on the graph structured data.

back-propagation. There has been some work on scene or scene related classification using methods that are or resemble graph networks. All of it is quite recent work, and none is yet done on the ADE dataset, to the best of our knowledge. Dornadula et al. [6] focus on few-shot scene graph prediction. They use an image as input and output a set of relations in the form of a subject, predicate and object. Mylavarapu et al. [7] use a Multi Relational Graph Convolutional Network (MRGCN) to model on-road vehicle behaviours from a sequence of temporally ordered frames as grabbed by a moving monocular camera. Chen et al. [8] combine local (convolutional) and global features (graph-based) to do an attention-based prediction of a scene based on an image. None of these methods were specifically suited for the dataset and/or our use case, because either the output was not a scene prediction or moving images were used.

2.1. Graph Neural Networks

Graph Neural Networks are a relatively new type of Deep Learning networks and operate on graphs. A graph contains nodes (v) (which represent entities), edges (e) (which represent relations) and possibly a global attribute (u) (which represents a property of the whole graph). The nodes contain attributes; a vector representing their data. The edges could contain attributes, but this is not required. Information is propagated through the graph, based on connectivity between nodes, to calculate an embedded representation of the attributes (see Figure 2). These representations can be used to make predictions on different levels: node-level, edge-level and graph-level. Node-level is often used for regression and classification, such as reasoning about physical systems. Edge-level is often used for classification and link prediction, such as predictions about interactions among entities. Graph-level is used for higher-level classification, such as predict the potential energy of a physical system or in our case scenes.

Currently a few overview papers are already present [9] [10] [11]. There are several ways to categorize the different types of GNNs, of which the categorization into model architecture is the most relevant, since it determines the way information is processed. Wu et al. [11] distinguish four categories of GNN models: Recurrent Graph Neural Networks (RecGNNs), Convolutional Graph Neural Networks (ConvGNNs), Graph Autoencoders (GAEs) and Spatial-

temporal Graph Neural Networks (STGNNs). In this paper, a ConvGNN is used, which updates node attributes based on their neighborhood and are commonly used in supervised learning settings.

The ConvGNNs translate the CNNs to the graph domain. They stack multiple graph neural layers to extract the high-level node representation and aggregate their own features and their neighbors' features. The ConvGNNs use a fixed number of layers, but have different weights in each layer. There are two ways to exploit the convolutional properties; via spectral-based or spatial-based approaches.

2.1.1. Spatial Methods

Spatial-based approaches use convolutions directly on the graph, by operating in the spatial neighborhood of a node. Within the spatial approaches, there are several approaches. Message Passing Neural Networks (MPNNs) use a general framework of spatial-based ConvGNNs. Graph Isomorphic Networks (GINs) [12] tackle the problem that many ConvGNN approaches cannot distinguish different graph structures based on the graph embedding. With isomorphic information, this is possible. Graph Attention Networks (GATs) [13] use attention mechanisms to learn the relative weights between two nodes. In the comparison by Wu et al. [11] it is stated that spatial methods are preferred over spectral methods because of efficiency, generality and flexibility.

2.1.2. Spectral Methods

Spectral-based approaches apply the graph convolution in the Fourier domain, by using the eigenvalues of the graph Laplacian. They are founded in the signal processing field.

Kipf et al. [14] proposed a Graph Convolutional Network (GCN) for learning on graph-structured data and the task of node classification. They propose a first-order approximation of the operations in the Fourier domain and their GCN is state-of-the-art in Graph Networks. Node features are updated in hidden layers, using convolutional approximation. Every hidden layer has its own weights and nodes are updated based on their own features and their neighbors'. In the loss only the labeled nodes are taken into account.

2.1.3. Graph Network blocks

Battaglia et al. [15] propose a general Graph Network framework (GN block) that generalizes various approaches for networks that operate on graphs. The framework takes a graph as input, performs computations and returns a graph as output. A graph is represented by nodes (V), edges (E) and a global attribute (u). The GN block contains update functions ϕ to update the attributes in the graph and aggregation functions ρ to aggregate information edges to nodes and from edges and nodes to the global attribute (see Figure 3). The update functions can be non-neural functions, however, in practice most networks do use neural layers.

GN-blocks are a generalization of all kinds of GNN models and allow the usage of node, edge and global attributes. They support all kind of aggegration and update functions. GCNs can be seen as a specific implementation of a GN block; they use the node updates, which they implemented as a convolutional function. GCNs are designed for node classification, where

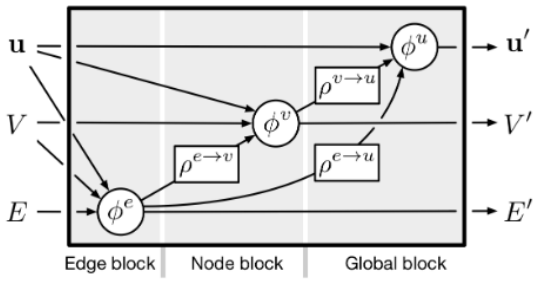

Figure 3: Full GN block as proposed by Deepmind (directly taken from [15].

the flexibility of the GN blocks allows the usage of edge and global features as well and can, therefore, be used for graph classification.

Because scene classification is about classifying the whole graph into a scene category, the GN-block with neural layers is implemented as our Graph Neural Network.

3. Experimental Setup

3.1. Data & Knowledge

As explained in the introduction, first an object detector is applied on the input images, resulting in images with detections, similar to those of Figure 4. The object label and the number of occurrences per label of all detected objects with a confidence of 0.1 or higher (see the Introduction for a motivation), on average 42 detections per image, are the data-driven input of the graph.

The dataset used in the experiments is the ADE Scene dataset [3], of which a subset is used for the experiments (see Table 1). This dataset is state of the art for scene classification and contains images, object segmentations with labels and a scene label per image. Four scene classes are selected for the experiments: *Living Room*, *Office*, *Home Office*, *Dorm Room*. These classes are selected based on our use case, and the scenes are similar enough to make it not too easy to detect.

Based on the ADE-given training and validation set, two splits for the training set and the test set are created. Table 1 shows the number of samples per set used in the splits. With these splits, three dataset splits are created (named as training split-test split):

- Small-Small: Training set of 24 samples, to reflect the open world use case. Test set as given by the original ADE validation set.
- Small-Big: Training set of 24 samples. Test set expanded with (a selection of) the unused training samples to create a bigger (balanced) test set.
- Full-Small: Using the original ADE training and test set.

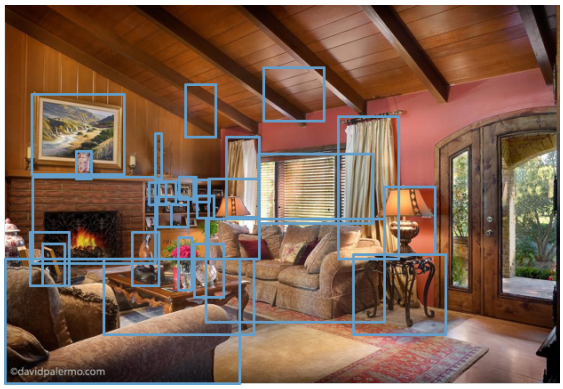

Figure 4: Object Detections on an image from the ADE dataset.

Scene	Small Training	Full Training	Small Test	Big Test
Dorm room	24	61	6	41
Living room	24	697	70	100
Office	24	112	11	98
Home office	24	95	10	81

Table 1
Number of samples in the dataset splits.

3.1.1. Creating Knowledge

One of the most important factors in a hybrid AI system is the knowledge. Preferably this knowledge is complementary to the data, and therefore, add information that increases the performance. World knowledge or expert knowledge is preferably used. However, since the automatically trained object detectors are mimicked by a pretrained object detector, the available knowledge is bounded towards these classes. This means only 80 object classes, of which many are unrelated to our scene classification, can be linked in the knowledge. On the one hand, this scopes the knowledge, on the other hand only a few known relations can be exploited as knowledge.

The expert knowledge in the experiments is based on the statistics of the training set, instead of crafting the knowledge ourselves. Using the statistics as the knowledge, resulted in the best knowledge for this dataset, however, might not be the most general knowledge. The assumption is that especially in the dataset split with little training samples, the statistics could be used to help the network to learn the statistics using the knowledge. The average number of object occurrences per scene is counted and added to as a knowledge-indicator for that scene if the average is above 0.7. The object 'book' was the only exception, since it appeared in all scenes, but much more in office (5) and home office (8). The knowledge about object-indicators for a scene is used as 'evidence' for that scene. The knowledge is used as a strong indication for a scene if indicator-objects are detected.

The following scene-indicator knowledge is used:
Living Room: couch, potted plant, dining table, vase,
Office: keyboard, book,
Home Office: tv, book,
Dorm Room: bed, bottle.

chair	bed	couch	tv
3	2	1	1

chair	bed	couch	tv	living room	dorm room	office	home office
3	2	1	1	1	2	0	1

Figure 5: Simplified example of the input for the MLP Data only (left) and MLP Data + Know (right).

3.2. Methods

This section describes the different methods that are compared in the results. The baselines are the first four methods, and our proposed method is a GNN with a data-knowledge combined input graph (GNN Data + Know). Ablation studies are used to determine the best parameter settings, such as batch size, learning rate and number of epochs. Due to the limited space, these results are not included in this paper. The epoch selection is done by choosing the best number of epochs after a run of 2000 epochs, based on the loss on a validation-split (20%) of the training set.

3.2.1. Knowledge only

The first baseline is Knowledge only, in which the evidence for each of the scenes is counted using the knowledge. This method is, thus, not learned but inferred. The detected occurrences of the different objects are counted and the scene with the highest knowledge-based evidence is chosen as the predicted scene.

3.2.2. MLP Data only

In the MLP Data only experiment, the table with object occurrences is fed to a 3 layer neural network (MLP) (for an example see Figure 5 left). The dense layers have a size of 64 and a RELU activation. The final layer contains a Softmax activation. The learning rate for the small training set is set to 0.005 with a RMSprop optimizer and binary cross-entropy loss, the batch size is 32 and the number of epochs is set to 50. The network is implemented using Keras [16]. For the full training set the learning rate is set to 0.0001 and the number of epochs is set to 100.

3.2.3. MLP Data + Know

This MLP has the same model as the previous section, but the evidence from the *Knowledge only* is used as an additional columns in the input (one per scene; all objects detected as evidence are summed in the column, see Figure 5, right). The learning rate for the small training set is set to 0.005, the batch size is 32 and the number of epochs is set to 12. For the full training set the learning rate is set to 0.0001 and the number of epochs is set to 25.

3.2.4. GNN Data only

For the GNN, we base our code on Pytorch Geometric [17]. The dataset is converted in a way that each image is represented as a graph. Within each graph, each node represents a detected object. Each node has a N-HotEncoding node attribute, encoding two types of information: the name of the object and the number of occurrences of that object (N). Objects that are not detected in the image, are not part of the graph. All nodes are connected to all other nodes,

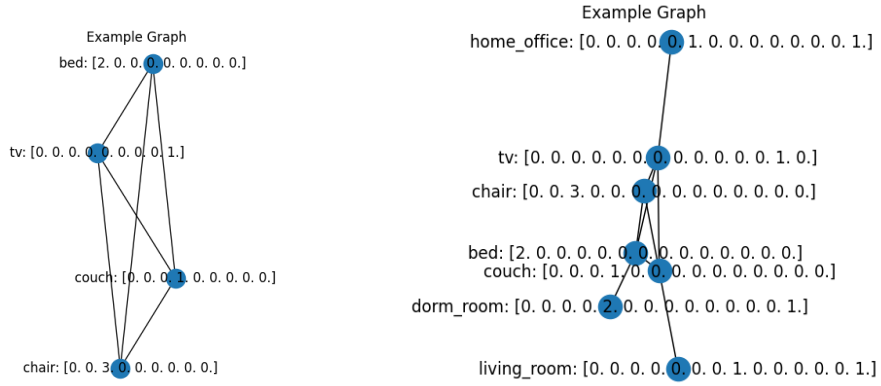

Figure 6: Simplified example of an input graph in GNN Data only (left) and GNN Data + Know (right) is used as input.

representing that they all occur in the same image. An example of these graphs is shown in Figure 6 (left). In that figure 3 chairs, 1 couch, 1 tv and 2 beds are detected. The scene label is predicted by the GNN model and represented by a global attribute of length four (one for every scene labels), which is initialized with zeros.

The GNN model used for classification, is based on the general GN model structure from Deepmind as described by Battaglia et al. [15] (see Figure 3). Since our graph does not contain edge attributes, we only constructed a Node block and a Global block. Our GNN model design is shown in Figure 7. The N-HotEncoded node attributes are passed through a MLP (Multi-Layer Perceptron) to obtain updated node attributes. In order to apply aggregation of the neighbors, the mean of the updated neighbors and the original node attributes are concatenated and passed through another MLP. This provides updated node attributes. The updated node attributes of all nodes are averaged and used together with the global attribute itself to update the global attribute in another MLP. This whole cycle of updating the node attributes and global attribute, is repeated for 3 three times. This is comparable to message passing of three times with shared weights. Sharing the weights reduces the number of parameters to train, while still allowing for the information to flow through the graph structure. The three single MLP modules consists of 2 layers of 32 nodes. Between the two layers a ReLU activation function is used.

The updated global attributes are used to determine the loss. The cross-entropy loss is used after applying a softmax to the output of the global attributes. The learning rate for the small training set is set to 0.005, the batch size is 16 and the number of epochs is set to 125. For the full training set the learning rate is also set to 0.005 and the number of epochs is set to 150.

3.2.5. GNN Data + Know

In the GNN model with knowledge, the knowledge is added as a node to the graph and an indicator whether the node is a knowledge or a data node. This means that the N-HotEncoding is, in the case of our 4 scenes, 5 longer. For each image, if there is evidence for a certain scene, a node with the scene name and the total amount of evidence is added. An edge between the evidence-object and the scene is created. This means that for the knowledge, the graphs are not fully connected any more, as can be seen in Figure 6 (right). Compared to the graph on the

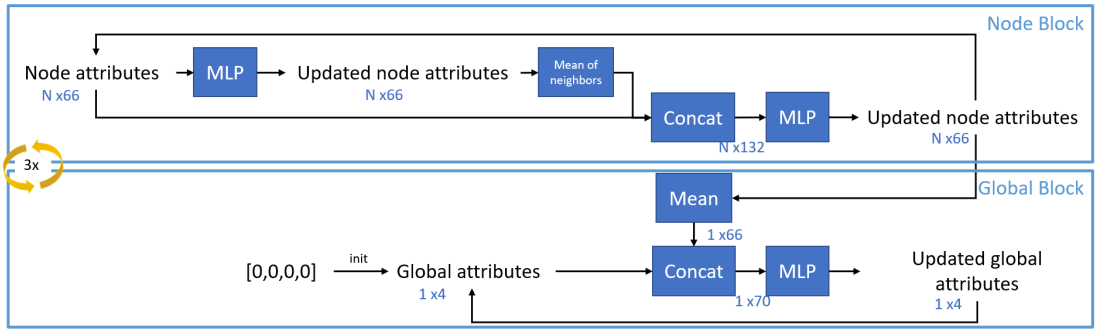

Figure 7: Overview of the GNN model containing a Node block and a Global block.

left, evidence for a living room (1), home office (1) and dorm room (2) is added.

The same model as in the GNN Data only experiment is used. The learning rate for the small training set is set to 0.001, the batch size is 16 and the number of epochs is set to 200. For the full training set the learning rate is set to 0.005 and the number of epochs is set to 175.

3.3. Evaluation Metric

We use accuracy as evaluation metric, defined as the number of correct classifications divided by the total number of samples. We run the MLP and GNN 5 times and report the mean accuracy. We also calculate the confusion matrix to gain insight in the mistakes made.

4. Results

Table 2 shows the results of the methods on the different dataset splits. The results show that for the Small-Small dataset the performance of the MLP is lower with knowledge (MLP Data + Know) compared to without knowledge (Data only), whereas the performance of the GNN increases with knowledge. Furthermore, the performance of the GNN with knowledge is higher than all other models, including the Knowledge only. The results on the dataset Small-Big are slightly different. Both the MLP and the GNN benefit from using knowledge, but performance is not higher compared to the Knowledge only baseline. The results on the dataset Full-Small show that when a lot of training data is present, the increase of performance with knowledge is not better compared to using the data only (for MLP). The GNN is also not able to outperform the MLP methods.

Table 3 shows performance per scene for the Small-Small dataset split (most related to our use case), which is part of the confusion matrix (full matrix is omitted due to space limitations). It shows that the type of mistakes differ, based on the source of information used. For example, the knowledge model works well for the *living room*, but bad for the *dorm room*. Both the MLP and GNN data only work well for the *dorm room*. Combining the data and knowledge in the GNN outperforms both the data only and knowledge only for *dorm room* (all models) and *office* (compared to only GNN). For *home office* and *living room*, performance of GNN Data + Know is similar to knowledge only, whereas with the MLP performance is only similar or drops.

	Small-Small		Small-Big		Full-Small	
Model	Mean	StDev	Mean	StDev	Mean	StDev
Knowledge only	0.58		0.58		0.59	
MLP Data only	0.63	0.06	0.53	0.01	0.86	0.01
MLP Data + Know	0.50	0.09	0.57	0.03	0.77	0.02
GNN Data only	0.51	0.13	0.46	0.08	0.52	0.03
GNN Data + Know	0.65	0.07	0.56	0.02	0.69	0.10

Table 2
Mean Accuracy for the models on the different datasets

Model	Dorm room	Home office	Living Room	Office
Knowledge only	0.4	0.5	0.75	0.36
MLP Data only	0.8	0.34	0.68	0.51
MLP Data + Know	0.76	0.34	0.58	0.36
GNN Data only	0.72	0.24	0.57	0.18
GNN Data + Know	0.84	0.5	0.72	0.52

Table 3
Mean accuracy per scene on dataset Small-Small

5. Discussion

In the results, the potential of using GNNs as well as the potential of adding knowledge is shown. For the Small-Small dataset, both GNN and knowledge are adding value. In the Small-Big experiment the knowledge is adding value to methods, but not better than using knowledge only. For the Full - Small experiment the performance on the data only for the MLP is good and the GNN cannot outperform the MLP data only. In this discussion, we want to critically discuss these results.

First, results are only shown on one task and data splits based on one dataset. From this dataset only 4 scenes are chosen. The performance has not been tested on other scenes, with other (or more) object detectors, nor on other datasets, which limits the conclusions about GNNs and knowledge to this specific setting.

Second, the knowledge currently used, is very limited. Only a few object classes are used as knowledge, and no normalization towards the number of indicator-objects per scene, is applied. This could have influenced the performance, because the biggest class also had the most knowledge objects. The way of choosing, representing and using knowledge could, thus, be improved and experimented with.

Third, with the current implementation is seems that the GNN can differentiate between the sources (data and knowledge) and benefit from the information, but there is still room for improvement. According to the literature, graph networks profit from graph-structured data and the choice for all-to-all combinations in the graphs could limit the ability to learn from it. More information could be added to the graphs when, for example, a hierarchy based on object types is put into the graph structure or spatial information is added. Although we belief that our GNN implementation is suited for our use case, other GNN models can also be suited and performance should be compared.

Finally it should be noticed that not every type of knowledge and data can be represented by a graph structure and, therefore, the application is limited to specific domains. In order to construct a graph there should be relational information in the data and the knowledge should be linked to that in a relational way. Ideally these relations add new information to the data.

6. Conclusion and Future Work

This paper covers a use case in which novel scenes have to be detected using noisy concept detectors. For that use case, we propose to use a GNN because it can handle input with a different length, and we propose to combine data and knowledge within the GNN.

Three different splits on the ADE dataset are used to compare performance of methods using knowledge only, data only and knowledge and data for both a MLP and a GNN. The results show that the GNN with knowledge can outperform all other models, but only in specific settings. If the performance of the data only is already high, it is hard to gain even higher performance by adding the knowledge. This is often the case in situations where there is lot of training data available. When there is only limited data available, adding knowledge could improve performance in specific settings. If the knowledge and the data are not complementary, it is hard to improve performance.

In future research, we would like to improve the GNN by for example using hierarchical information and making more use of the graph structure. We want to explore the combination of GNNs with knowledge on other datasets, such as one without images, to get further insight in which cases this combination is benificial and in which not. We would like to integrate our data and knowledge based predictions with work that is currently performed on user-specific explainability and experiment whether the added knowledge improves the interpretation of the explanations. Additionally, we would like to explore the usage of knowledge in more traditional ML approaches such as a naive Bayes classifier or Latent Dirichlet Allocation (LDA), either as inspiration for our method or as an additional baseline. Finally, we would like to further explore the different GNN types. For example use the GCN convolutional properties for the graph classification setting or use Graph Attention Networks. Learning with attention would enhance the possibilities for explaining the predictions.

Acknowledgments

We would like to thank Albert Huizing, Gertjan Burghouts and Joris Sijs for their valuable feedback, and the ERP Hybrid AI project from TNO for their financial support.

References

[1] G. Beckers, J. Sijs, J. van Diggelen, R. J. van Dijk, H. Bouma, M. Lomme, R. Hommes, F. Hillerstrom, J. van der Waa, A. van Velsen, et al., Intelligent autonomous vehicles with an extendable knowledge base and meaningful human control, in: Counterterrorism, Crime Fighting, Forensics, and Surveillance Technologies III, volume 11166, International Society for Optics and Photonics, 2019, p. 111660C.

[2] GluonCV, Gluoncv toolkit, https://cv.gluon.ai/, 2020.
[3] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, A. Torralba, Scene parsing through ade20k dataset, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 633–641.
[4] Y. Bengio, Learning deep architectures for AI, Now Publishers Inc, 2009.
[5] W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural computation 29 (2017) 2352–2449.
[6] A. Dornadula, A. Narcomey, R. Krishna, M. Bernstein, F.-F. Li, Visual relationships as functions: Enabling few-shot scene graph prediction, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2019, pp. 0–0.
[7] S. Mylavarapu, M. Sandhu, P. Vijayan, K. M. Krishna, B. Ravindran, A. Namboodiri, Understanding dynamic scenes using graph convolution networks, arXiv preprint arXiv:2005.04437 (2020).
[8] X. Chen, L.-J. Li, L. Fei-Fei, A. Gupta, Iterative visual reasoning beyond convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7239–7248.
[9] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, arXiv preprint arXiv:1812.08434 (2018).
[10] D. Bacciu, F. Errica, A. Micheli, M. Podda, A gentle introduction to deep learning for graphs, Neural Networks (2020).
[11] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S. Y. Philip, A comprehensive survey on graph neural networks, IEEE Transactions on Neural Networks and Learning Systems (2020).
[12] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, arXiv preprint arXiv:1810.00826 (2018).
[13] P. Velickovic, W. Fedus, W. L. Hamilton, P. Lio, Y. Bengio, R. D. Hjelm, Deep graph infomax., in: ICLR (Poster), 2019.
[14] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).
[15] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261 (2018).
[16] Keras, Keras toolkit, https://keras.io/, 2020.
[17] M. Fey, J. E. Lenssen, Fast graph representation learning with PyTorch Geometric, in: ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.