Improving Multi-Label Image Classification with Graphs Neural Networks

Théo VIEL
MVA - ENPC
theo.viel.enpc@gmail.com
Supervisor : Michal Valko

Abstract

Multi-label image classification is a wide-spread problematic that aims at predicting which objects are present in an image. Later improvements correlate with those in single-label image classification, but such methods omit to capture the relationships between objects. It appears that some are more likely to be found together, therefore graphs of relationships between labels can be leveraged to improve the usual classification pipeline. Therefore, we review three graph neural networks based methods that use this additional knowledge to improve the performances of the models. The studied approaches propose modifications to the widely spread Convolutional Neural Networks, that improve the results on usual benchmarks.

1 Introduction

Multi-label image classification aims at classifying the presence of a given set of objects in an image. It extends the well known single-label image classification tasks where only one of the classes can be found at a time. Therefore, the problem is usually more complicated, but also more interesting. The most common approach is to independently learn the probability of appearance of each item using Convolutional Neural Networks (CNN) (e.g. ResNet[6], DenseNet[7] and Inception [15] networks). As-is, those methods are a strict generalisation of the single-label classification one, and progress in multi-label classification usually come from progresses in single-label classification.

What those models fail to use and capture is the complex relationships between objects. In fact, modelling the probability of appearance of a label independently from others is not adapted to the multi-label image classification task. The co-occurrence of objects is what makes the multi-label task harder, and using the dependencies between labels is one of the key to tackle it. Indeed, some objects are known to very rarely appear together (e.g. horses and planes), whereas some are frequently found together (e.g. chairs and tables). Such intuition can be learned by the model by leveraging the graph structure of objects interactions.

Given a labelled dataset, it is straightforward to obtain the co-occurrences of the classes, and therefore model the probability of an object appearing given the other. The approach is then to learn label-dependant features, and propagate them by message passing using a graph weighted by these probabilities. Graph Neural Networks (GNN) are particularly adapted as they can be used alongside with a CNN : parameters of both networks can be jointly learned using back-propagation. This general idea has been used in different ways, which we will study here.

The first approach [13], is to replace the usual classification layer of a CNN with a Graph Gated Neural Network (GGNN) [11] which allows for the classifier to include object relationships. This architecture also permits the use of larger graphs, which will contain more information than the one obtained from the co-occurrences in the dataset. The reasoning is that that humans learn faster using word associations, and therefore a neural network could benefit from using large knowledge graphs.

Graphs in Machine Learning Project, MVA 2019.

Similarly, [1] introduces a semantic interaction module : a GGNN using the conditional probabilities of label appearances in the data. In addition, the authors propose a semantic decoupling module, which is an attention mechanism using word embeddings, to guide the model toward areas concerning specific classes. The association of these two layers enables to associate each part of the image to a specific concept (label or cluster of labels), and then propagate the detected information between related classes of objects. These two modules constitute the Semantic-Specific Graph Representation Learning (SSGRL) framework.

The approach proposed in [2] differs on the fact that the GNN now aims at learning classifiers instead of image features. The task of extracting image features is left to the CNN, and they use Graph Convolutional Networks (GCN) [9] to learn semantic-aware classifiers. GCNs aims at mapping word-embeddings in the space of classifiers, using once again the appearance conditional probability matrix for message passing.

The main contributions of this work are the following :

- We first introduce the notions needed to understand how GNNs can be adapted to the multi-label classification task.
- We then re-implement and benchmark the proposed methods on the Pascal VOC Dataset [5] and show that they indeed contribute to an improvement in results. The code will be documented and made publicly available on GitHub at `https://github.com/TheoViel/graphMLC`.
- We also conduct some experiments regarding some interesting points in the paper and check how merging some of the ideas affects the results.

2 Frameworks

2.1 Convolutional Neural Networks and Multi-Label Classification

Methods discussed here use CNNs to extract image level features. The CNN takes in input an image I and extracts a feature map $\mathbf{f}_{HW}^I$ of size $H \times W \times N$ where W is the width and H the height, i.e. we have a vector of $\mathbb{R}^N$ for specific areas of the image. This feature map can then be pooled to obtain a representation of the image $\mathbf{f}^I$ in $\mathbb{R}^N$.

Usually, this pooled vector is fed to a single dense layer with C units, where C is the number of classes, to obtain the probabilities of each class $\mathbf{p} \in [0,1]^C$. This enables the use of the binary cross entropy loss (BCE) to update the weights, given the ground truth $\mathbf{y} \in \{0,1\}^C$:

$$\mathcal{L}(\mathbf{p}, \mathbf{y}) = \sum_{c=1}^{C} \left(\mathbf{y}_c \log(\mathbf{p}_c) + (1 - \mathbf{y}_c) \log(1 - \mathbf{p}_c)\right) \tag{1}$$

Implicitly, a sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ is used to ensure probabilities are in $[0, 1]$. It is either used in the loss function or as activation on the last layer.

2.2 Graph Gated Neural Network

Introduced in [11], GGNNs adapt the Gated Recurrent Unit (GRU) [3] propagation model to the graph architecture. Given a graph with $|\mathcal{V}|$ nodes, at each iteration t, each node v has a representation $h_v^{(t)}$. For simplicity, we will consider an oriented graph with 1 type of edges, but the framework adapts to graphs with more edge labels. The graph is the represented by a matrix $\mathbf{A} = [\mathbf{A}^{(in)}, \mathbf{A}^{(out)}] \in \mathbb{R}^{|\mathcal{V}| \times 2|\mathcal{V}|}$ where $\mathbf{A}^{(in)}$ (resp. $\mathbf{A}^{(out)}$) stores the incoming (resp outgoing) edges of each node. $\mathbf{A}_v$ refers to the two columns of $\mathbf{A}$ corresponding to node v. At each time step t and for all v, the update

is the following :

$$\mathbf{a}_v^{(t)} = \mathbf{A}_v^T \left[\mathbf{h}_1^{(t-1)} ... \mathbf{h}_{|\mathcal{V}|}^{(t-1)}\right]^T + \mathbf{b} \quad \text{(Message Passing)} \tag{2}$$

$$\mathbf{z}_v^{(t)} = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \quad \text{(Update Gate)} \tag{3}$$

$$\mathbf{r}_v^{(t)} = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \quad \text{(Reset Gate)} \tag{4}$$

$$\tilde{\mathbf{h}}_v^{(t)} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^{(t)} \odot \mathbf{h}_v^{(t-1)}\right)\right) \tag{5}$$

$$\mathbf{h}_v^{(t)} = \left(1 - \mathbf{z}_v^{(t)}\right) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^{(t)} \odot \tilde{\mathbf{h}}_v^{(t)} \tag{6}$$

Where σ is the sigmoid function and $\odot$ is the element-wise product. Note that the message passing step gives in $\mathbf{a}_v^{(t)} \in \mathbb{R}^{2|\mathcal{V}|}$ where its $|\mathcal{V}|$ first (resp. last) components are the sum of the $(\mathbf{h}_v^{(t-1)})_{1 \leq v \leq |\mathcal{V}|}$ weighted by the component of $\mathbf{A}_v$ corresponding to $\mathbf{A}^{(in)}$ (resp. $\mathbf{A}^{(out)}$). Noting s the size of the node features of the GGNN, the learned parameters of the GGNN are : $\mathbf{b} \in \mathbb{R}^{2s}$ the bias during message passing, $\mathbf{W}, \mathbf{W}^z, \mathbf{W}^r \in \mathbb{R}^{2s \times s}$ and $\mathbf{U}, \mathbf{U}^z, \mathbf{U}^r \in \mathbb{R}^{s \times s}$.

Inspired from the ideas of [13], we propose to feed the pooled features $\mathbf{f}^I$ to a linear layer of size $|\mathcal{V}|$ whose outputs will be used as $\mathbf{h}^{(0)}$. $\mathbf{h}^{(0)}$ is repeated to be of shape $|\mathcal{V}| \times s$, enabling the network to learn features of dimension greater than 1. The final layer is a linear one of size C which takes as input the concatenation of $\mathbf{h}^{(0)}$ before repetition, and $\mathbf{h}^{(T)}$ the output of the GGNN after T steps flattened to a vector. Note that $|\mathcal{V}|$ does not have to be equal to C, which enables the use of more precise knowledge graphs, and therefore allows for a significant performance boost. The framework is represented figure 1.

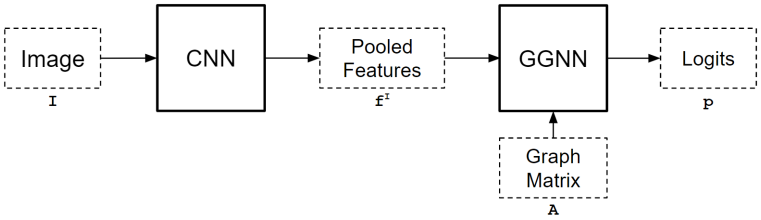


Figure 1: GGNN for image classification framework. We omit the dense layers after and before the GGNN for clarity.

2.3 Semantic-Specific Graph Representation Learning

2.3.1 Semantic decoupling layer

The semantic decoupling (SD) layer is a word embedding guided attention mechanism that takes the feature maps $\mathbf{f}_{HW}^I$ as input. Let us denote $\mathbf{X}_c \in \mathbb{R}^E$ the embedding of the label corresponding to class c. We omit the I in the notation for clarity. For each class c and location (h, w), we compute the attention weights the following way :

$$\tilde{\mathbf{f}}_{hw}^c = \mathbf{P}^T \tanh\left(\mathbf{U}^\mathbf{T}\mathbf{f}_{hw} \odot \mathbf{V}^\mathbf{T}\mathbf{X}_c\right) + \mathbf{b} \tag{7}$$

$$\tilde{\mathbf{a}}_{hw}^c = \tilde{\mathbf{f}}_{hw}^c \cdot \mathbf{w} + c \quad \text{(Attention weights)} \tag{8}$$

$$\mathbf{a}_{hw}^c = \frac{\exp(\tilde{\mathbf{a}}_{hw}^c)}{\sum_{h',w'} \exp(\tilde{\mathbf{a}}_{h'w'}^c)} \quad \text{(Softmax)} \tag{9}$$

The semantic decoupled features $\mathbf{f}^c, 1 \leq c \leq C$ are obtained by weighted sum over all the locations:

$$\mathbf{f}^c = \sum_{h,w} \mathbf{a}_{hw}^c \mathbf{f}_{hw}^c \tag{10}$$

$\mathbf{U} \in \mathbb{R}^{N \times d}$ maps the image features in the hidden space and $\mathbf{V} \in \mathbb{R}^{E \times d}$ maps the word embeddings in the hidden space. $\mathbf{P} \in \mathbb{R}^{d \times d}$ and $\mathbf{b} \in \mathbb{R}^d$ the associated bias is a dense layer of the term-wise

product of the two mappings. $\mathbf{w} \in \mathbb{R}^d$ enables the computation of the attention weights, $c \in \mathbb{R}$ being the associated bias. d is the hidden dimension of the attention mechanism and the only hyper-parameter here.

2.3.2 Semantic interaction layer

The semantic interaction (SI) layer consists of a GGNN which takes as input the class-specific features $\mathbf{f}^c, 1 \leq c \leq C$ extracted by the semantic decoupling layer. The matrix $\mathbf{A}$ is based on the probability of label i appearing given that label j is in the image, which is estimated using the same data as for training. The final output of the network uses C 1-unit dense layers. Each of them takes as input the the outputs of both the semantic decoupling and interaction layers of the associated class, and basically outputs the probability of the class being present in the image.

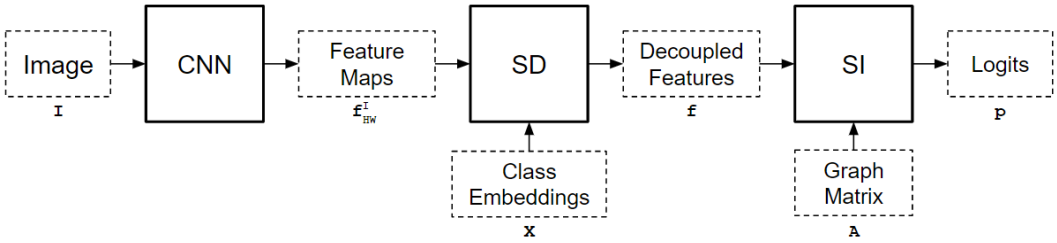


Figure 2: Illustration of the SSGRL framework. The image is first fed in the CNN, from which we extract the feature maps $\mathbf{f}^I_{HW}$, which are fed into the semantic interaction module. The output features $\mathbf{f}$ are fed into the semantic interaction module. We feed both the outputs of the SD and SI modules to the final dense layers (not represented here) to obtain the logits.

2.4 Graph Convolutional Network

Introduced in [9], GCNs aim at learning node-level features $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d}$ using message passing. A GCN layer can be written the following way :

$$\mathbf{H}^o = h(\mathbf{A}\mathbf{H}^i\mathbf{W}) \tag{11}$$

The i and o exponents refer to input and output parameters, h is the activation function and $\mathbf{W} \in \mathbb{R}^{d^i \times d^o}$ are trainable weights. $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the adjacency matrix as previously seen.

The idea introduced in [2] is to use the graph structure to replace the classification layer plugged on the CNN. It aims at learning them by starting from word embedding based label representations, to which we apply several GCNs. GCNs transform class embeddings $\mathbf{X}_c \in \mathbb{R}^E$ to classifiers $\mathbf{W}_c \in \mathbb{R}^N$, and the probability of class c is obtained the following way :

$$p_c = \mathbf{W}_c \cdot \mathbf{f}^I + b_c \tag{12}$$

$\mathbf{f}^I \in \mathbb{R}^N$ is the pooled image features, $b_c \in \mathbb{R}$ is the bias associated to class c. The framework is illustrated figure 3

[2] also highlights the need to cleverly build the adjacency matrix. They start by using the conditional appearance probability as before, but add two post-processing steps :

$$\mathbf{A'}_{ij} = \begin{cases} 1 & \text{if } \mathbf{A}_{ij} > \tau \\ 0 & \text{if } \mathbf{A}_{ij} \leq \tau \end{cases} \qquad \text{(Thresholding)} \tag{13}$$

$$\mathbf{A''}_{ij} = \begin{cases} p \Big/ \sum_{j=1, j \neq i} \mathbf{A'}_{ij} & \text{if } i \neq j \\ 1 - p & \text{if } i = j \end{cases} \qquad \text{(Re-weighting)} \tag{14}$$

The goal of the thresholding step is to smooth noisy edges, and the goal of the re-weighting one is to prevent over-smoothing. Over-smoothing occurs because of the binary correlation matrix : similar classes may become indistinguishable after the GCNs, as the thresholding matrix will pass the message the same ways for nodes in the same cluster.

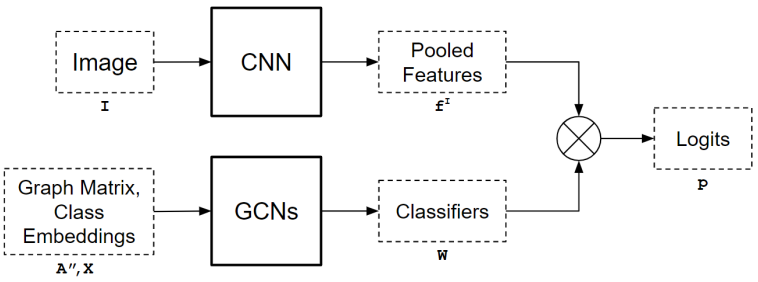


Figure 3: Illustration of the GCN framework.

3 Experiments

3.1 Data and baseline

To benchmark the proposed methods, we use the Pascal Visual Object Classes Challenge (VOC 2007) dataset [5]. The first reason being its relatively small size (less than 10 000 images, half of them being for testing), which enables to run experiments in a reasonable time. The second one being that it is quite popular, and two of the studied methods ([2, 1]) provide benchmarks. However, the goal is not to reproduce the results stated in the paper, but to show that the proposed methods indeed boost the results. The evaluation metric is the mean average precision (mAP) over the 20 classes of the dataset. We use both train and val set for training and report results on the test set.

To conduct experiments, we will use the ResNet [6] architectures pretrained on ImageNet [4]. Motivated by lower run times, the lighter ResNet-34 will be used for most studies, unless otherwise specified. [2, 1] use ResNet-101 as a reference, therefore this architecture will be also considered. To push results a bit further, we will take a look at the performances of the 32x8d ResNeXt-101 [16] with weights pretrained using weakly supervised learning [12]. For the 101 layers architectures, only the last layers will be fine-tuned. Also, most experiments will be made using an image size of 224 but better results are achieved using larger image sizes such as 576 in [1]. The trade-off being that smaller batch sizes are needed and once again run times are longer. Baseline results are provided table 1. The pooled output is of size $N = 512$ for ResNet-34, and $N = 2048$ for ResNet-101 and ResNeXt-101.

Table 1: Baseline benchmarks.

Backbone	Image size	Run time	mAP
ResNet-34	224	2 mins	85.5
	576	14 mins	90.1
ResNet-101	224	3 mins	88.0
	576	20 mins	93.5
ResNeXt-101	224	7 mins	92.4
	576	36 mins	95.4

All the models are trained with the Adam [8] optimiser using the same two steps setup :

1. First the backbone will kept frozen, the last layers will be trained with a learning rate decreasing from 10^{-3} to 10^{-4} using a cosine scheduling. The idea is that we first only want to modify weights that are not pretrained. The batch size is fixed at 32. Most architectures require only 3 epochs to converge here, except for the GCN one which requires 10.
2. We then train the overall network for 5 additional epochs with a learning rate decreasing from 10^{-4} to 10^{-6} with a cosine scheduling. The batch size is 32 for an image size of 224, 8 otherwise (because of memory constraints).

The idea of using this pipeline is that it is robust enough to changes of architectures. Spending time tweaking hyper-parameters not related to the subject is not intended here. Regarding augmentations during train time, we use horizontal flips, random rotations of angle lower than 45 degrees, and some

adjustments on contrast, gamma and brightness. Note that crops are not used, because cropping out a label from an image can confuse the models. During test time, only the mandatory resizing transformation is kept.

3.2 Graph Gated Neural Network

[13] aims at adapting GGNN to image tasks, using a potentially big knowledge graph of label relationships. In fact, GGNN does not scale very well to big graphs (> 1000 nodes) according to the authors. However, given that we use a dataset with only 20 class, we can focus on relatively small graphs and therefore stick to the GGNN approach. In fact, table 2 justifies that we do not need huge graphs nor a high number of updates T. Experiments also showed that $s = 10$ was a good choice.

Table 2: Influence of $|\mathcal{V}|$ and T.

$\|\mathcal{V}\|$	100	100	100	300	500	1000
T	2	3	4	3	3	3
mAP	86.4	86.5	86.3	**86.6**	86.6	86.5

Provided that huge graphs are not needed, we also simplify a bit the scheme to construct the knowledge graph. We only use the Visual Genome [10] dataset, which contains a large enough number of relationships of the form $i \xrightarrow{r} j$ to build a fairly robust graph for our problem. We use the $R = 100$ most present relationships r and therefore construct R graphs :

$$\begin{cases} \mathbf{A}_{ij}^{(in)_r} = 1 & \text{if the relationship } i \xrightarrow{r} j \text{ is in the dataset} \\ \mathbf{A}_{ij}^{(out)_r} = 1 & \text{if the relationship } j \xrightarrow{r} i \text{ is in the dataset} \end{cases}$$

$\mathbf{A}^{(in)}$ and $\mathbf{A}^{(out)}$ are obtained by averaging over r. Performances for image size 224 are reported table 3, and we notice that the proposed approach improves the results by approximately 1 point on average. Using matrix the processing from [2] with $p = \tau = 0.1$ on our matrix $\mathbf{A}$, the mAP can be further improved, as reported under the **GGNN + $\mathbf{A}''$** column.

Table 3: Performance of the proposed GGNN approach (mAP).

Backbone	**Baseline**	**GGNN**	**GGNN + $\mathbf{A}''$**
ResNet-34	85.5	86.6	86.8
ResNet-101	88.0	88.6	88.7
ResNeXt-101	92.4	93.2	93.5

3.3 Semantic-Specific Graph Representation Learning

We follow the reasoning of [1] and justify the interests of using both of the proposed modules. Best performances were obtained using the GloVe [14] embeddings of size $E = 300$, hidden sizes of $d = 1024$ for the semantic decoupling module, and $T = 3$ for the semantic interaction one. Once again, we report the results of image size 224 and observe similar improvements as for the GGNN approach. As shown table 4, the proposed framework helps improving the results. Improvements are especially noticeable for the ResNet-34 model, with a 1.6% boost, but are almost zero for the ResNet-101 architecture.

Table 4: Performance of the proposed semantic interaction and semantic decoupling modules (mAP).

Backbone	**Baseline**	**SD**	**SD + SI**
ResNet-34	85.5	86.8	87.1
ResNet-101	88.0	88.0	88.1
ResNeXt-101	92.4	93.3	93.1

It appears that the ablative study provided by the authors shows much more convincing results. To ensure of the correctness of the proposed implementation, I use the same setting as the authors. They use an image size of 576 and the ResNet-101 backbone. Performances are reported in table 5 and prove that our implementation works fine. The difference in the two setups comes mainly from the transformations used. On train time, [1] uses random crops on images resized to 640, of size chosen randomly in $\{320, 384, 512, 576, 640\}$, and then resized to 576, the goal being to learn size-invariant features. Centred crops of size 576 are used on test time. We notice that the used setup here performs better overall, but that it reduces the contributions of the implemented modules.

Table 5: Comparison of our implementation with the author's one (mAP).

Setup	**ResNet101**	**ResNet101 + SD**	**ResNet101 + SD + SI**	**[1]**
Used in [1]	92.5	93.0	93.3	93.4
Proposed	93.5	93.3	**93.7**	n/a

As the semantic decoupling layer uses an attention mechanism over different zones of the image, we can extract the weights. This allows us to capture what information is learned by the model, and to make sure that the module works fine. An example on an image of the test set is shown figure 4, high attention weights correspond to the bright zones of the image. We use 576 as image size, as smaller images result in a coarser attention map.

(a) Class *Dog* (b) Class *Person*

Figure 4: Interpolated attention maps of the SD module.

3.4 Graph Convolutional Network

Once again, we benchmark the proposed method. We perform some hyper-parameter optimisation using the ResNet-34 architecture and an image size of 224, and once satisfying results are obtained, we move on to larger architectures. Experiments results on the GCN layers and matrix are reported table 6, we therefore went for $p = 0.25$, $\tau = 0.25$ and use 3 GCNs, the last one having N units, and the others $N/2$ (N being the size of the pooled output of the CNN). As in [2], using LeakyReLU [17] with a 0.2 slope as non-linearity in the GCNs yields better results.

Table 6: Influence of τ, p and number of GCNs.

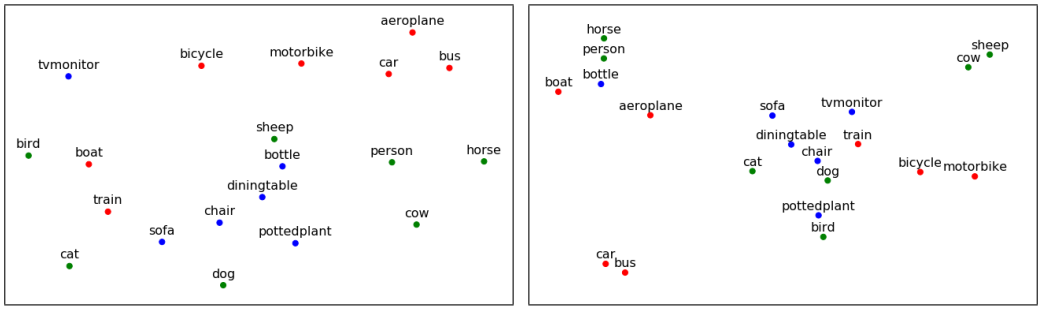
p	0.1	0.25	0.4	0.5	0.25	0.25	0.25	0.25
τ	0.25	0.25	0.25	0.25	0.1	0.5	0.25	0.25
n_{GCN}	2	2	2	2	2	2	3	1
mAP	86.5	**86.6**	86.5	85.0	86.5	86.4	86.2	85.0

Results for image size 224 are reported table 7 and once again the proposed approach proves to be useful. Indeed, on the one hand, adding the GCN layers contributes to a $\approx 0.5\%$ increase for both ResNet-34 and ResNeXt-101 architectures. The thresholding and re-weighting steps are effective as well and provide a $\approx 0.5\%$ boost to the approach with the non-modified $\mathbf{A}$ matrix.

Table 7: Performance of the GCNs (mAP).

Backbone	Baseline	GCN + A	GCN + A''
ResNet-34	85.5	86.1	86.6
ResNet-101	88.0	87.5	88.1
ResNeXt-101	92.4	93.1	93.4

The authors of [2] also highlight that the advantage of this approach is that the classifiers learned preserve the semantic structure of the classes, whereas the usual classifier weights do not depend on their meaning. The authors state that the learned classifier do not show any meaningful topology, but as shown figure 5a, this is not the case. The representation is less precise than the one learned by the GCN classifiers 5b, but the visual features learned by the CNN provides enough information to have a meaningful embedding.



(a) Baseline classifiers

(b) GCN classifiers

Figure 5: t-SNE plot of the learned classifiers. Class clusters were coloured the following way for clearer visualisation : *red = vehicles, blue = objects, green = animals*.

Table 8: Per class average precision of the models. A ResNet-34 backbone and a 576 image size is used. All three methods improve the baseline mAP, with GGNN outperforming the others by a substantial margin.

	plane	*bike*	*bird*	*boat*	*bottle*	*bus*	*car*	*cat*	*chair*	*cow*	*table*	*dog*	*horse*	*moto.*	*person*	*plant*	*sheep*	*sofa*	*train*	*tv*	**mAP**
base.	98.6	95.4	96.2	95.3	70.3	89.8	94.6	95.8	78.7	88.5	81.4	94.6	96.3	93.1	98.2	76.9	90.8	78.6	98.2	89.7	90.0
ggnn	**99.5**	95.9	**97.5**	**97.2**	**76.3**	**93.0**	**95.9**	**97.0**	**81.1**	94.3	82.3	96.3	**97.6**	**94.4**	**98.5**	80.0	**96.1**	78.8	**98.5**	**92.4**	**92.1**
ssgrl	98.9	**96.7**	97.1	96.1	74.8	92.5	95.8	96.4	79.5	**94.5**	81.8	**96.5**	96.6	94.0	**98.5**	**80.9**	92.4	**80.0**	97.7	90.4	91.5
gcn	98.5	95.9	97.2	95.7	74.8	91.4	95.2	**97.0**	78.7	92.2	**82.4**	96.3	96.4	93.3	**98.5**	80.4	93.1	78.9	98.1	91.6	91.3

4 Conclusion

In this work, we presented three state of the art approaches to multi-label classification, namely Graph Gated Neural Networks, the Semantic-Specific Graph Representation Learning framework, and Graph Convolutional Networks. Every single one of them contributes to a gain in performances on the Pascal VOC 2007 dataset. Capturing label relationships appears to play a significant role the multi-label classification problem and such methods very likely extend to other tasks. For instance, some experiments were conducted on a multi-task learning problem on text data, but first results were not convincing enough to be reported here. In fact, the idea of using the co-occurrence of objects can be generalised to any task by measuring the relationship between two targets, paving the way to Graph Neural Network approaches.

References

[1] Tianshui Chen, Muxin Xu, Xiaolu Hui, Hefeng Wu, and Liang Lin. Learning semantic-specific graph representation for multi-label image recognition. 08 2019.

[2] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. Multi-label image recognition with graph convolutional networks. *CoRR*, abs/1904.03582, 2019.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[9] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[10] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016.

[11] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *Proceedings of ICLR'16*, April 2016.

[12] Dhruv Kumar Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018.

[13] Kenneth Marino, Ruslan Salakhutdinov, and Abhinav Gupta. The more you know: Using knowledge graphs for image classification. *CoRR*, abs/1612.04844, 2016.

[14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[16] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[17] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.