Mapping Images to Scene Graphs with Permutation-Invariant Structured Prediction

Roei Herzig[*]
Department of Computer Science
Tel Aviv University
roeiherzig@mail.tau.ac.il

Moshiko Raboh[*]
Department of Computer Science
Tel Aviv University
shikorab@gmail.com

Gal Chechik
Department of Computer Science
Google Brain, Bar-Ilan University
gal.chechik@gmail.com

Jonathan Berant
Department of Computer Science
Tel Aviv University
joberant@cs.tau.ac.il

Amir Globerson
Department of Computer Science
Tel Aviv University
amir.globerson@tau.ac.il

Abstract

Machine understanding of complex images is a key goal of artificial intelligence. One challenge underlying this task is that visual scenes contain multiple inter-related objects, and that global context plays an important role in interpreting the scene. A natural modeling framework for capturing such effects is structured prediction, which optimizes over complex labels, while modeling within-label interactions. However, it is unclear what principles should guide the design of a structured prediction model that utilizes the power of deep learning components. Here we propose a design principle for such architectures that follows from a natural requirement of permutation invariance. We prove a necessary and sufficient characterization for architectures that follow this invariance, and discuss its implication on model design. Finally, we show that the resulting model achieves new state of the art results on the Visual Genome scene graph labeling benchmark, outperforming all recent approaches.

1 Introduction

Understanding the semantics of a complex visual scene is a fundamental problem in machine perception. It requires recognizing multiple objects in a scene, together with their spatial and functional relations. This set of objects and relations is sometimes represented as a graph, connecting objects (nodes) with their relations (edges) and is known as a *scene graph* (Figure 1). Scene graphs provide a compact representation of the semantics of an image, and can be useful for semantic-level interpretation and reasoning about a visual scene Johnson et al. (2018). Scene-graph prediction is the problem of inferring the joint set of objects and their relations in a visual scene.

Importantly, objects and their relations are interdependent. Thus, inferring these from an image benefits greatly from predicting them jointly. This joint prediction is a special case of a more general problem, inferring multiple inter-dependent labels, which is the research focus of the field of

[*]Equal Contribution.

Preprint. Work in progress.

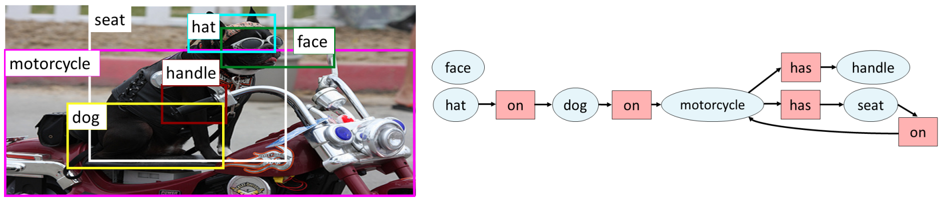


Figure 1: An image and its scene graph from the Visual Genome dataset (Krishna et al., 2017). The scene graph captures the entities in the image (nodes, blue circles) like *dog* and their relations (edges, red circles) like $\langle$hat, on, dog$\rangle$.

structured prediction. Structured prediction has attracted considerable attention because it applies to many learning problems and poses unique theoretical and applied challenges (e.g., see Belanger et al., 2017; Chen et al., 2015; Taskar et al., 2004). It is therefore a natural approach for predicting scene graphs from images.

Structured prediction models typically define a score function $s(x, y)$ that quantifies how well a label assignment y is compatible with an input x. In the case of understanding complex visual scenes, x is an image, and y is a complex label containing the labels of objects detected in an image and the labels of their relations. In this setup, the *inference task* amounts to finding the label that maximizes the compatibility score $y^* = \arg\max_y s(x, y)$. This score-based approach separates a scoring component – implemented by a parametric model, from an optimization component – aimed at finding a label that maximizes that score. Unfortunately, for a general scoring function $s(\cdot)$, the space of possible label assignments grows exponentially with input size. For instance, the set of possible object label assignments over a scene graph is too large even for relatively simple images, since the vocabulary of candidate objects may contain thousands of objects. Thus, inferring the label assignment that maximizes a scoring function is computationally hard in the general case.

An alternative approach to score-based methods is to map an input x to a structured output y with a “black box" neural network, without explicitly defining a score function. This raises a natural question: what is the right architecture for such a network? Here we take an axiomatic approach and argue that one important property such networks should satisfy is invariance to a particular type of input permutation. We then prove that this invariance is equivalent to imposing certain structural constraints on the architecture of the network, and describe architectures that satisfy these constraints.

To evaluate our approach, we first demonstrate on a synthetic dataset that respecting permutation invariance is important, as models of comparable size that violate this invariance require more training data. Then, we tackle the problem of scene graph generation. We describe a model that satisfies the permutation invariance property, and show that it achieves state-of-the-art results on the competitive Visual Genome benchmark (Krishna et al., 2017), demonstrating the power of our new design principle.

In summary, the novel contributions of this paper are: a) Deriving sufficient and necessary conditions for respecting graph-permutation invariance in deep structured prediction architectures. b) Empirically demonstrating the benefit of graph-permutation invariance. c) Developing a state-of-the-art model for scene graph prediction on a large dataset of complex visual scenes.

2 Structured prediction

Scored-based methods in structured prediction define a function $s(x, y)$ that reflects the degree to which y is compatible with x, and infer a label by maximizing $s(x, y)$ (e.g., see Belanger et al., 2017; Chen et al., 2015; Lafferty et al., 2001; Meshi et al., 2010; Taskar et al., 2004). Most score functions previously used decompose as a sum over *simpler* functions, $s(x, y) = \sum_i f_i(x, y)$, where solving $\max_y f_i(x, y)$ can be performed efficiently. This local maximization forms the basic building block of algorithms for approximately maximizing $s(x, y)$. One way to achieve this is to restrict $f_i(x, y)$ to depend only on a small subset of the y variables.

The renewed interest in deep learning led to efforts to integrate deep networks with structured prediction, including modeling the f_i functions as deep networks. In this context, the most widely-used score functions are singleton $f_i(y_i, x)$ and pairwise $f_{ij}(y_i, y_j, x)$. Initial work used a two-stage architecture, learning local scores independently of the structured prediction goal (Chen et al., 2014; Farabet et al., 2013). Later works considered *end-to-end* architectures where the inference algorithm is part of the computation graph (Chen et al., 2015; Pei et al., 2015; Schwing & Urtasun, 2015; Zheng et al., 2015). Recent works also go beyond pairwise scores, modelling global factors as well (Belanger et al., 2017; Gygli et al., 2017).

Score-based methods provide several advantages. First, they allow intuitive specification of local dependencies between labels and how these translate to global dependencies. Second, for linear score functions, the learning problem has natural convex surrogates Lafferty et al. (2001); Taskar et al. (2004). Third, inference in large label spaces is sometimes possible via exact algorithms or empirically accurate approximations. However, with the advent of deep scoring functions $s(x, y; \boldsymbol{w})$, learning is no longer convex. Thus, it is worthwhile to rethink the architecture of structured prediction models, and consider models that map inputs x to outputs y directly without explicitly maximizing a score function. We want these models to enjoy the expressivity and predictive power of neural networks, while maintaining the ability to specify local dependencies between labels in a flexible manner. In the next section, we present such an approach and consider a natural question: what should be the properties of such a deep neural network used for structured prediction.

3 Permutation-Invariant Structured Prediction

We begin with some notation, focusing on structures with pairwise interactions, as these are simpler in terms of notation, and sufficient for describing the structure in many problems. We denote a structured label by $y = [y_1, \dots, y_n]$. In a score-based approach, the score is defined via a set of singleton scores $f_i(y_i, x)$ and pairwise scores $f_{ij}(y_i, y_j, x)$, where the overall score $s(x, y)$ is the sum of these scores. For brevity, we denote $f_{ij} = f_{ij}(y_i, y_j, x)$ and $f_i = f_i(y_i, x)$. An inference algorithm takes as input the local scores f_i, f_{ij} and outputs an assignment that maximizes $s(x, y)$. We can thus view inference as a black-box that takes node and edge-dependent inputs (i.e., the scores f_i, f_{ij}) and returns a label y, even without an explicit score function $s(x, y)$. While numerous inference algorithms exist for this setup, including belief propagation (BP) and mean field, here we develop a framework for a deep labeling algorithm (we avoid the term "inference" since the algorithm does not explicitly maximize a score function). Such an algorithm will be a black-box with the f functions as input and the labels $y_1, \dots, y_n$ as output. We next ask what architecture such an algorithm should have.

We follow with several definitions. A *graph labeling function* $\mathcal{F} : (V, E) \to Y$ is a function whose input is an ordered set of node features $V = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_n]$ and an ordered set of edge features $E = [\boldsymbol{z}_{1,2} \dots, \boldsymbol{z}_{i,j}, \dots, \boldsymbol{z}_{n,n-1}]$. E.g., $\boldsymbol{z}_i$ can be the array of values f_i, and $\boldsymbol{z}_{i,j}$ can be the table of values $f_{i,j}$. Assume $\boldsymbol{z}_i \in \mathbb{R}^d$ and $\boldsymbol{z}_{i,j} \in \mathbb{R}^e$. The output of $\mathcal{F}$ is a set of node labels $\boldsymbol{y} = [y_1, \dots, y_n]$. Thus, algorithms like BP are graph labeling functions. However, graph labeling functions need not correspond to any inference algorithm (i.e., an algorithm maximizing a score function). We denote the joint set of node and edge features by $\boldsymbol{z}$ (i.e., a set of $n + n(n-1) = n^2$ vectors). In Section 3.1 we discuss extensions to the case where only a subset of the edges is available).

A natural requirement is that the algorithm produces the same result when given the same score function. E.g., consider a label space with three variables y_1, y_2, y_3, and assume that the inference algorithm takes as input $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \boldsymbol{z}_{12}, \boldsymbol{z}_{13}, \boldsymbol{z}_{23}) = (f_1, f_2, f_3, f_{12}, f_{13}, f_{23})$, and outputs a label $\boldsymbol{y} = (y_1^*, y_2^*, y_3^*)$. When the algorithm is given an input that is permuted in a consistent way, $\boldsymbol{z}' = (f_2, f_1, f_3, f_{21}, f_{23}, f_{13})$, this defines *exactly* the same score function. Hence, the output should be $\boldsymbol{y} = (y_2^*, y_1^*, y_3^*)$. Most inference algorithms, including BP and mean field, satisfy this symmetry requirement by design. Here our goal is to design a deep learning black-box, and hence would like to guarantee invariance to input permutations. A black-box that violates this invariance wastes capacity on learning it at training time, which increases sample complexity, as shown in Sec. 5.1.

We next consider what happens to a graph labeling function, when node features are permuted by a permutation σ. Importantly, edge features are also permuted in a way that is consistent with σ.

Definition 1. *Let $\boldsymbol{z}$ be a set of node and edge features. Given a permutation σ of $\{1, \dots, n\}$, let $\sigma(\boldsymbol{z})$ be a new set of node and edge features given by $[\sigma(\boldsymbol{z})]_i = \boldsymbol{z}_{\sigma(i)}$ and $[\sigma(\boldsymbol{z})]_{i,j} = \boldsymbol{z}_{\sigma(i),\sigma(j)}$.*

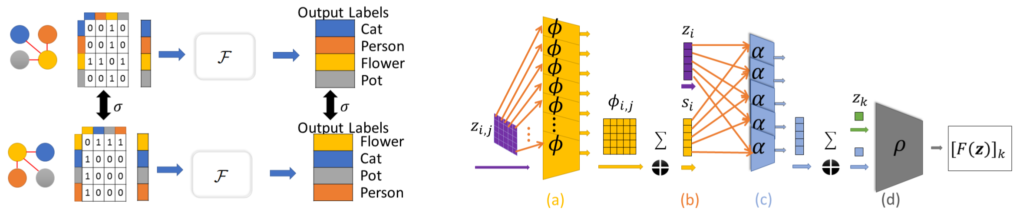


Figure 2: **Left**: Graph permutation invariance. A graph labeling function $\mathcal{F}$ is graph permutation invariant (GPI) if permuting the node features maintains the output. **Right**: a schematic representation of the GPI architecture in Theorem 1. Singleton features $\boldsymbol{z}_i$ are omitted for simplicity. First, the features $\boldsymbol{z}_{i,j}$ are processed element-wise by ϕ. Next, they are summed to create a vector $\boldsymbol{s}_i$, which is concatenated with $\boldsymbol{z}_i$. Third, a representation of the entire graph is created by applying $\boldsymbol{\alpha}$ n times and summing the created vector. The graph representation is then finally processed by ρ together with $\boldsymbol{z}_k$.

In what follows, we use the notation $\sigma([y_1, \dots, y_n]) = [y_{\sigma(1)}, \dots, y_{\sigma(n)}]$, namely, σ applied to a set of labels yields the same labels, only permuted by σ. We now provide our key definition of a function whose output is invariant to permutations of the input. See Figure 2 (left).

Definition 2. *A graph labeling function $\mathcal{F}$ is said to be* **graph-permutation invariant** *(GPI), if for all permutations σ of $\{1, \dots, n\}$ and for all $\boldsymbol{z}$ it satisfies: $\mathcal{F}(\sigma(\boldsymbol{z})) = \sigma(\mathcal{F}(\boldsymbol{z}))$.*

3.1 Characterizing Permutation Invariance

Motivated by the above discussion, we ask: what structure is necessary and sufficient to guarantee that $\mathcal{F}$ is GPI? Note that a function $\mathcal{F}$ takes as input an **ordered** set $\boldsymbol{z}$. Therefore its output on $\boldsymbol{z}$ could certainly differ from its output on $\sigma(\boldsymbol{z})$. To achieve permutation invariance, $\mathcal{F}$ should contain certain symmetries. E.g., one permutation invariant architecture is to define $y_i = g(\boldsymbol{z}_i)$ for any function g, but this is too restrictive to cover all permutation invariant functions. The next Theorem provides a complete characterization, while Figure 2 shows the corresponding architecture.

Theorem 1. *Let $\mathcal{F}$ be a graph labeling function. Then $\mathcal{F}$ is graph-permutation invariant if and only if there exist functions $\boldsymbol{\alpha}, \rho, \boldsymbol{\phi}$ such that for all $k = 1, \dots, n$:*

$$[\mathcal{F}(\boldsymbol{z})]_k = \rho(\boldsymbol{z}_k, \sum_{i=1}^{n} \boldsymbol{\alpha}(\boldsymbol{z}_i, \sum_{j \neq i} \boldsymbol{\phi}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j))), \tag{1}$$

where $\boldsymbol{\phi} : \mathbb{R}^{2d+e} \to \mathbb{R}^L$, $\boldsymbol{\alpha} : \mathbb{R}^{d+L} \to \mathbb{R}^W$ and $\boldsymbol{\rho} : \mathbb{R}^{W+d} \to \mathbb{R}$.

Proof. First, we show that any $\mathcal{F}$ satisfying the conditions of Theorem 1 is GPI. Namely, for any permutation σ, $[\mathcal{F}(\sigma(\boldsymbol{z}))]_k = [\mathcal{F}(\boldsymbol{z})]_{\sigma(k)}$. To see this, write $[\mathcal{F}(\sigma(\boldsymbol{z}))]_k$ using Eq. 1 and Definition 1:

$$\rho(\boldsymbol{z}_{\sigma(k)}, \sum_{i} \boldsymbol{\alpha}(\boldsymbol{z}_{\sigma(i)}, \sum_{j \neq i} \boldsymbol{\phi}(\boldsymbol{z}_{\sigma(i)}, \boldsymbol{z}_{\sigma(i),\sigma(j)}, \boldsymbol{z}_{\sigma(j)}))). \tag{2}$$

The second argument of ρ above is invariant under σ, because the sum considers an index i and all other indices j, hence the same elements only under permutation. The expression therefore equals to:

$$\rho(\boldsymbol{z}_{\sigma(k)}, \sum_{i} \boldsymbol{\alpha}(\boldsymbol{z}_i, \sum_{j \neq i} \boldsymbol{\phi}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j))) = [\mathcal{F}(\boldsymbol{z})]_{\sigma(k)}$$

where equality follows from Eq. 1. We thus proved that Eq. 1 implies graph permutation invariance.

Next, we prove any black-box GPI function can be expressed as in Eq. 1. Namely, we show how to define $\boldsymbol{\phi}, \boldsymbol{\alpha}$ and ρ that can implement any permutation invariant function $\mathcal{F}$. The key idea is to construct $\boldsymbol{\phi}, \boldsymbol{\alpha}$ such that the second argument of ρ in Eq. 1 contains all the information about the graph features $\boldsymbol{z}$, including the edges they originated from . Then, the function ρ consists of an application of the black box $\mathcal{F}$ to this representation, followed by extracting the label y_k. To simplify notation assume edge features are scalar ($e = 1$). The extension to vectors is simple, but involves more indexing. We also assume $\boldsymbol{z}_k$ uniquely identifies the node (i.e., no two nodes share the same node feature), which can be achieved by adding the index as another feature of $\boldsymbol{z}_k$. Finally, we assume $\mathcal{F}$ is a function only of the pairwise features $\boldsymbol{z}_{i,j}$, which can be achieved by adding singleton features into the pairwise ones.

Let H be a hash function with L buckets mapping node features $\boldsymbol{z}_i$ to an index (bucket). Assume that H is perfect (this can be achieved for a large enough L). Define $\boldsymbol{\phi}$ to map the pairwise features to a vector of size L. Let $\mathbb{1}[j]$ be a one-hot vector of dimension $\mathbb{R}^L$, with one in the j^{th} coordinate. Recall that we consider scalar $\boldsymbol{z}_{i,j}$ so that $\boldsymbol{\phi}$ is indeed in $\mathbb{R}^L$, and define $\boldsymbol{\phi}$ as: $\boldsymbol{\phi}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j) = \mathbb{1}[H(\boldsymbol{z}_j)]\, z_{i,j}$, i.e., $\boldsymbol{\phi}$ "stores" $z_{i,j}$ in the unique bucket for node j.

Let $\boldsymbol{s}_i = \sum_{\boldsymbol{z}_{i,j} \in E} \boldsymbol{\phi}(\boldsymbol{z}_i, z_{i,j}, \boldsymbol{z}_j)$ be the second argument of $\boldsymbol{\alpha}$ in Eq. 1 ($\boldsymbol{s}_i \in \mathbb{R}^L$). Then, since all $\boldsymbol{z}_j$ are distinct, $\boldsymbol{s}_i$ stores all the pairwise features for neighbors of i in unique positions within its L coordinates. Since $\boldsymbol{s}_i(H(\boldsymbol{z}_k))$ contains the feature $z_{i,k}$ whereas $\boldsymbol{s}_j(H(\boldsymbol{z}_k))$ contains the feature $\boldsymbol{z}_{j,k}$, we cannot simply sum the $\boldsymbol{s}_i$, since we would lose the information of which edges the features originated from. Instead, we define $\boldsymbol{\alpha}$ to map $\boldsymbol{s}_i$ to $\mathbb{R}^{L \times L}$ such that each feature is mapped to a distinct location. Formally:

$$\boldsymbol{\alpha}(\boldsymbol{z}_i, \boldsymbol{s}_i) = \mathbb{1}[H(\boldsymbol{z}_i)]\, \boldsymbol{s}_i^T \, . \tag{3}$$

$\boldsymbol{\alpha}$ outputs a matrix that is all zeros except for the features corresponding to node i that are stored in row $H(\boldsymbol{z}_i)$. The matrix $M = \sum_i \boldsymbol{\alpha}(\boldsymbol{z}_i, \boldsymbol{s}_i)$ (namely, the second argument of ρ in Eq. 1) is a matrix with all the edge features in the graph including the graph structure.

To complete the construction we set ρ to have the same outcome as $\mathcal{F}$. We first discard rows and columns in M that do not correspond to original nodes (reducing M to dimension $n \times n$). Then, we use the reduced matrix as the input $\boldsymbol{z}$ to the given $\mathcal{F}$. Assume for simplicity that M does not need to be contracted (this merely introduces another indexing step). Let the output of $\mathcal{F}$ on M be $\boldsymbol{y} = y_1, \dots, y_n$. Then we set $\rho(\boldsymbol{z}_k, M) = \boldsymbol{y}_{H(\boldsymbol{z}_k)}$. Since $\mathcal{F}$ is invariant to permutations this indeed returns the output of $\mathcal{F}$ on the original input.

$\square$

Extension to general graphs So far, we discussed complete graphs, where edges correspond to valid feature pairs. However, many graphs of interest might be incomplete. For example, an n-variable chain graph in sequence labeling has only $n-1$ edges. For such graphs, the input to $\mathcal{F}$ would not contain all $\boldsymbol{z}_{i,j}$ pairs but rather only features corresponding to valid edges of the graph, and we are only interested in invariances that preserve the graph structure, namely, the automorphisms of the graph. Thus, the desired invariance is that $\sigma(\mathcal{F}(\boldsymbol{z})) = \mathcal{F}(\sigma(\boldsymbol{z}))$, where σ is not an arbitrary permutation but an automorphism. It is easy to see that a simple variant of Theorem 1 holds in this case. All we need to do is replace in Eq. 2 the sum $\sum_{j \neq i}$ with $\sum_{j \in N(i)}$, where $N(i)$ are the neighbors of node i in the graph. The arguments are then similar to the proof above.

Implications of Theorem 1 Our result has interesting implications for deep structured prediction. First, it highlights that the fact that the architecture "collects" information from different edges of the graph, in an invariant fashion via the $\boldsymbol{\alpha}, \boldsymbol{\phi}$ functions. In other words, these functions provide a summary of the graph that is sufficient for downstream algorithms. Second, the architecture is parallelizable, as all $\boldsymbol{\phi}$ functions can be applied simultaneously. This is in contrast to recurrent models Zellers et al. (2017) which are harder to parallelize and are thus slower in practice. Last, the theorem suggests several common architectural structures that can be used within GPI. We briefly mention two of these. 1) **Attention:** Attention is a powerful component in deep learning architectures (Bahdanau et al., 2015), but most inference algorithms do not use attention. Intuitively, in attention each node i aggregates features of neighbors through a weighted sum, where the weight is a function of the neighbor's relevance. For example, the label of an entity in an image may depend more strongly on entities that are spatially closer. Attention can be naturally implemented in our GPI characterization, and we provide a full derivation for this implementation in the appendix. It plays a key role in our scene graph model described below. 2) **RNNs:** Because GPI functions are closed under composition, for any GPI function $\mathcal{F}$ we can run $\mathcal{F}$ iteratively by providing the output of one step of $\mathcal{F}$ as part of the input to the next step and maintain GPI. This results in a recurrent architecture, which we use in our scene graph model.

4 Related Work

The concept of architectural invariance was recently proposed in DEEPSETS (Zaheer et al., 2017). The invariance we consider is much less restrictive: the architecture does not need to be invariant to all permutations of singleton and pairwise features, just those consistent with a graph re-labeling. This characterization results in a substantially different set of possible architectures.

Deep structured prediction. There has been significant recent interest in extending deep learning to structured prediction tasks. Much of this work has been on semantic segmentation, where convolutional networks (Shelhamer et al., 2017) became a standard approach for obtaining "singleton scores" and various approaches were proposed for adding structure on top. Most of these approaches used variants of message passing algorithms, unrolled into a computation graph (Xu et al., 2017). Some studies parameterized parts of the message passing algorithm and learned its parameters (Lin et al., 2015). Recently, gradient descent has also been used for maximizing score functions (Belanger et al., 2017; Gygli et al., 2017). An alternative to deep structured prediction is greedy decoding, inferring each label at a time based on previous labels. This approach has been popular in sequence-based applications (e.g., parsing (Chen & Manning, 2014)), relying on the sequential structure of the input, where BiLSTMs are effectively applied. Another related line of work is applying deep learning to graph-based problems, such as TSP (Bello et al., 2016; Gilmer et al., 2017; Khalil et al., 2017). Clearly, the notion of graph invariance is important in these, as highlighted in (Gilmer et al., 2017). They however do not specify a general architecture that satisfies invariance as we do here, and in fact focus on message passing architectures, which we strictly generalize. Furthermore, our focus is on the more general problem of structured prediction, rather than specific graph-based optimization problems.

Scene graph prediction. Extracting scene graphs from images provides a semantic representation that can later be used for reasoning, question answering, and image retrieval (Johnson et al., 2015; Lu et al., 2016; Raposo et al., 2017). It is at the forefront of machine vision research, integrating challenges like object detection, action recognition and detection of human-object interactions (Liao et al., 2016; Plummer et al., 2017). Prior work on scene graph predictions used neural message passing algorithms (Xu et al., 2017) as well as prior knowledge in the form of word embeddings (Lu et al., 2016). Other work suggested to predict graphs directly from pixels in an end-to-end manner Newell & Deng (2017). NeuralMotif (Zellers et al., 2017), currently the state-of-the-art model for scene graph prediction on Visual Genome, employs an RNN that provides global context by reading sequentially the independent predictions for each entity and relation and then refines those predictions. The NEURALMOTIF model maintains GPI by fixing the order in which the RNN reads its inputs and thus only a single order is allowed. However, this fixed order is not guaranteed to be optimal. Conversely, our architecture is guaranteed to provide the same output for all valid input permutations.

5 Experimental Evaluation

We empirically evaluate the benefit of GPI architectures. First, using a synthetic graph-labeling task, and then, for the problem of mapping images to scene graphs.

5.1 Synthetic Graph Labeling

We start with studying GPI on a synthetic problem, defined as follows: Given an input graph $G = (V, E)$ where each node $i \in V$ belongs to a set $\Gamma(i) \in \{1, \dots, K\}$, the goal is to compute for each node the number of neighbors that belong to the same set, namely, the label of a node is $y_i = \sum_{j \in N(i)} \mathbb{1}[\Gamma(i) = \Gamma(j)]$. We generated random graphs with 10 nodes by sampling each edge independently and uniformly, and sampling a set $\Gamma(i) \in \{1, \dots, K\}$ for every node uniformly (larger graphs produced similar results). The node features $\boldsymbol{z}_i \in \{0,1\}^K$ are one-hot vectors of $\Gamma(i)$ and the edge features $\boldsymbol{z}_{i,j} \in \{0,1\}$ indicate whether $ij \in E$. We compare two standard non-GPI architectures and one GPI architecture: (a) A GPI-architecture for graph prediction, described in detail in Section 5.2. We used the basic version without attention and RNN. (b) LSTM: We replace $\sum \phi(\cdot)$ and $\sum \alpha(\cdot)$, which perform aggregation in Theorem 1 with two LSTMs with state size of 200 that read their input in random order. (c) A fully-connected (FC) feed-forward network with 2 hidden layers of 1000 nodes each. The input to the fully connected model is a concatenation of all node and

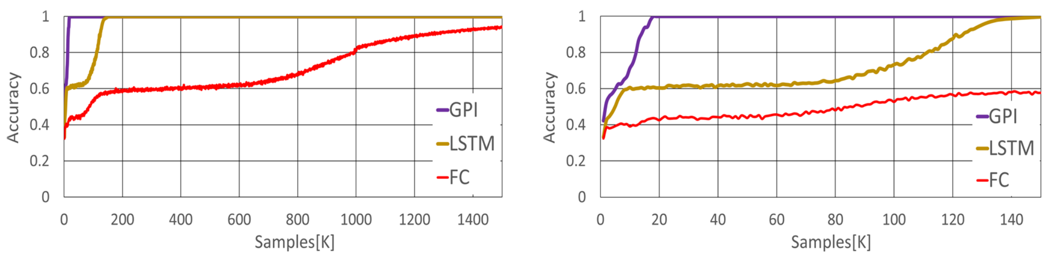


Figure 3: Accuracy as a function of sample size for graph labeling. Right is a zoomed in version of left.

pairwise features. The output is all node predictions. The focus of the experiment is to study sample complexity. Therefore, for a fair comparison, we use the same number of parameters for all models.

Results are shown in Figure 3, demonstrating that SGP requires far fewer samples to converge to the correct solution. This illustrates the advantage of an architecture with a proper inductive bias.

5.2 Scene-Graph Classification

We evaluate the GPI approach on the main task of this paper, inferring scene graphs from images (Figure 1). In this problem, the input is an image annotated with a set of *bounding boxes* for the entities in the image.[2] The goal is to label each bounding box with the correct entity category and every pair of entities with their relation, such that they form a coherent *scene graph*.

We begin by describing our *Scene Graph Predictor (SGP)* model. We aim to predict two types of variables. The first is entity variables $[y_1, \dots, y_n]$ for all bounding boxes. Each y_i can take one of L values (e.g., "dog", "man"). The second is relation variables $[y_{n+1}, \dots, y_{n^2}]$ for every pair of bounding boxes. Each such y_j can take one of R values (e.g., "on", "near"). Our graph connects variables that are expected to be inter-related. It contains two types of edges: 1) **entity-entity edge** connecting every two entity variables (y_i and y_j for $1 \leq i \neq j \leq n$. 2) **entity-relation edges** connecting every relation variable y_k (where $k > n$) to its two entity variables. Thus, our graph is not a complete graph and our goal is to design an architecture that will be invariant to any automorphism of the graph, such as permutations of the entity variables.

For features $\boldsymbol{z}$, we simply re-use features learned by the baseline model from Zellers et al. (2017) [3]. Specifically, for entity variables y_i we have a vector $\boldsymbol{z}_i \in \mathbb{R}^L$ that assigns probabilities to all entities appearing in y_i. We further augment $\boldsymbol{z}_i$ with bounding box coordinates and simple features of those. Similarly, for relation variables y_j we have a vector $\boldsymbol{z}_j \in \mathbb{R}^R$ that assigns probabilities to all possible relations between the two variables in the relation j. For entity-entity pairwise features $\boldsymbol{z}_{i,j}$ we again use the relation probability for this pair. Because the output of SGP are probability distributions over entities and relations, we use them as an the input $\boldsymbol{z}$ to SGP once again in a recurrent manner and maintain GPI. Finally, we also explored the use word embeddings. For each $\boldsymbol{z}_i$ above, we concatenate the embedding of the most probable entity and relation label.

We next describe the main components of the GPI architecture. First, we focus on the parts that output the entity labels. $\boldsymbol{\phi}_{ent}$ is the network that integrates features for two entity variables y_i and y_j. It simply takes $\boldsymbol{z}_i, \boldsymbol{z}_j, \boldsymbol{z}_{i,j}$ as input and outputs a vector of dimension n_1. Next, the network $\boldsymbol{\alpha}_{ent}$ takes as input the outputs of $\boldsymbol{\phi}_{ent}$ for all neighbors of an entity, and uses the attention mechanism described above to output a vector of dimension n_2. Finally, the ρ_{ent} network takes these n_2 dimensional vectors and outputs L logits predicting the entity value. The ρ_{rel} network takes as input the α_{ent} representation of the two entities, as well as $\boldsymbol{z}_{i,j}$ and transforms the output into R logits. See appendix for specific network architectures.

[2]For simplicity, we focus on the task where boxes are given.

[3]The baseline does not use any LSTM or context, and is thus unrelated to the main contribution of Zellers et al. (2017).

5.2.1 Experimental Setup and Results

Dataset. We evaluated our approach on Visual Genome (VG) (Krishna et al., 2017), a dataset with 108,077 images annotated with bounding boxes, entities and relations. On average, images have 12 entities and 7 relations per image. For a proper comparison with previous results (Xu et al., 2017; Newell & Deng, 2017; Zellers et al., 2017), we used the data from (Xu et al., 2017), including the train and test splits. For evaluation, we used the same 150 entities and 50 relations as in (Xu et al., 2017; Newell & Deng, 2017; Zellers et al., 2017). To tune hyper-parameters, we also split the training data into two by randomly selecting 5K examples, resulting in a final 70K/5K/32K split for train/validation/test sets.

Training. All networks were trained using Adam (Kingma & Ba, 2014) with batch size 20. Hyper-parameter values below were chosen based on the validation set. The SGP loss function was the sum of cross-entropy losses over all entities and relations in the image. In the loss, we penalized entities 4 times more strongly than relations, and penalized negative relations 10 times more weakly than positive relations.

Evaluation. (Xu et al., 2017) defined three different subtasks when inferring scene graphs, and we focus on two: **(1) SGCls:** Given ground-truth bounding boxes for entities, predict all entity categories and relations categories. **(2) PredCls:** Given bounding boxes annotated with entity labels, predict all relations. Following (Lu et al., 2016), we used Recall@K as the evaluation metric. It measures the fraction of correct ground-truth triplets that appear within the K most confident triplets proposed by the model. Two evaluation protocols are used in the literature, which differ by whether they enforce graph constraints over model predictions. The first *graph-constrained* protocol requires that the top-K triplets assign one consistent class per entity and relation. The second *unconstrained* protocol does not enforce any such constraints. We report results on both protocols, following (Zellers et al., 2017).

Models and baselines. We compare four variants of our GPI approach with the reported results of four baselines that are currently the state-of-the-art on various scene graph sub-tasks. All models use the same data split and pre-processing as (Xu et al., 2017): 1) (Lu et al., 2016): This work leverages word embeddings to fine-tune the likelihood of predicted relations. 2) (Xu et al., 2017): This model passes messages between entities and relations, and iteratively refines the feature map used for prediction. 3) (Newell & Deng, 2017): The PIXEL2GRAPH model uses associative embeddings (Newell et al., 2017) to produce a full graph from the image. 4) (Zellers et al., 2017): The NEURALMOTIF method encodes global context for capturing high-order motifs in scene graphs, and the BASELINE outputs the entities and relations distributions without using the global context. The following variants GPI were compared: 1) GPI: NO ATTENTION: Our GPI model, but with no attention mechanism. Instead, following Theorem 1, we simply sum the features. 2) GPI: NEIGHBORATTENTION: Our GPI model, with attention over neighbors features. 3) GPI: LINGUISTIC: Same as GPI: NEIGHBORATTENTION but also concatenating the word embedding vector, as described above.

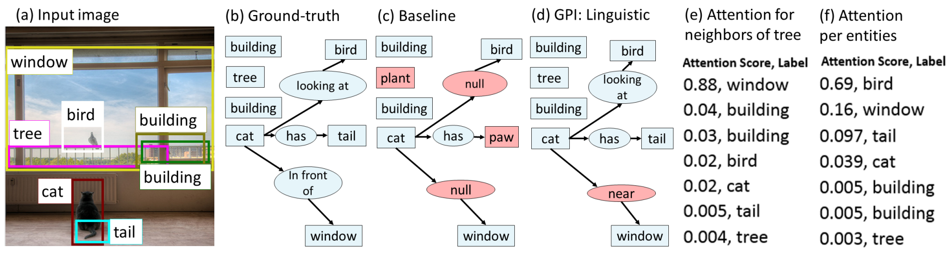
Results. Table 1 lists recall@50 and recall@100 for three variants of our approach compared with five baselines, evaluating with graph constraints. The GPI approach performs well, LINGUISTIC outperforms all baselines for SGCls and comparable to state of the art model for PredCls. Note that PredCl is an easier task, which makes less use of structure, and it's thus not surprising that our method and Zellers et al. (2017) have similar accuracy on it. Figure 4 illustrates the model behavior. Predicting isolated labels with z_i (4c) mislabels several entities, but these are corrected at the final output (4d). Figure 4e shows that the system learned to attend more to nearby entities (the window and building are closer to the tree), and 4f shows that stronger attention is learned for the classes bird, presumably because it is usually more informative than common classes like tree.

6 Conclusion

We presented a deep learning approach to structured prediction, which constrains the architecture to be invariant to structurally identical inputs. As in score-based methods, our approach relies on pairwise features, capable of describing inter-label correlations, and thus inheriting the intuitive aspect of score-based approaches. However, instead of maximizing a score function (which leads

	Constrained Evaluation				Unconstrained Evaluation			
	SGCls		PredCls		SGCls		PredCls	
	R@50	R@100	R@50	R@100	R@50	R@100	R@50	R@100
(Lu et al., 2016)	11.8	14.1	35.0	27.9	-	-	-	-
(Xu et al., 2017)	21.7	24.4	44.8	53.0	-	-	-	-
Pixel2Graph (Newell & Deng, 2017)	-	-	-	-	26.5	30.0	68.0	75.2
Neural Motifs (Zellers et al., 2017)	35.8	36.5	**65.2**	**67.1**	44.5	47.7	**81.1**	**88.3**
Baseline (Zellers et al., 2017)	34.6	35.3	63.7	65.6	43.4	46.6	78.8	85.9
No Attention	35.3	37.2	64.5	66.3	44.1	48.5	79.7	86.7
Neighbor Attention	35.7	38.5	64.6	66.6	44.7	49.9	80.0	87.1
Linguistic	**36.5**	**38.8**	65.1	66.9	**45.5**	**50.8**	80.8	88.2

Table 1: Test set results for graph-constrained evaluation (the returned triplets must be consistent with a scene graph) and for unconstrained evaluation (the returned triplets could be not consistent with a scene graph).



Attention Score, Label	Attention Score, Label
0.88, window	0.69, bird
0.04, building	0.16, window
0.03, building	0.097, tail
0.02, bird	0.039, cat
0.02, cat	0.005, building
0.005, tail	0.005, building
0.004, tree	0.003, tree

Figure 4: (a) An input image with bounding boxes from VG. (b) The ground-truth scene graph. (c) The Baseline fails to recognize some entities (*tail* and *tree*) and relations (*in front of* instead of *looking at*). (d) GPI:LINGUISTIC fixes most incorrect LP predictions. (e) *Window* is the most significant neighbor of *Tree*. (f) The entity *bird* receives substantial attention, while *Tree* and *building* are less informative.

to computationally-hard inference), we directly produce an output that is invariant to equivalent representations of the pairwise terms.

The axiomatic approach can be extended in many ways. For image labeling, geometric invariances (shift or rotation) may be desired. In other cases, invariance to feature permutations may be desirable. We leave the derivation of the corresponding architectures to future work. Finally, there may be cases where the invariant structure is unknown and should be discovered from data, which is related to work on lifting graphical models Bui et al. (2013). It would be interesting to explore algorithms that discover and use such symmetries for deep structured prediction.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Belanger, David, Yang, Bishan, and McCallum, Andrew. End-to-end learning for structured prediction energy networks. In Precup, Doina and Teh, Yee Whye (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 429–439. PMLR, 2017.

Bello, Irwan, Pham, Hieu, Le, Quoc V, Norouzi, Mohammad, and Bengio, Samy. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Bui, Hung Hai, Huynh, Tuyen N., and Riedel, Sebastian. Automorphism groups of graphical models and lifted variational inference. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, pp. 132–141, 2013.

Chen, Danqi and Manning, Christopher. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 740–750, 2014.

Chen, Liang Chieh, Papandreou, George, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *Proceedings of the Second International Conference on Learning Representations*, 2014.

Chen, Liang Chieh, Schwing, Alexander G, Yuille, Alan L, and Urtasun, Raquel. Learning deep structured models. In *Proc. ICML*, 2015.

Farabet, Clement, Couprie, Camille, Najman, Laurent, and LeCun, Yann. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1915–1929, 2013.

Gilmer, Justin, Schoenholz, Samuel S, Riley, Patrick F, Vinyals, Oriol, and Dahl, George E. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Gygli, Michael, Norouzi, Mohammad, and Angelova, Anelia. Deep value networks learn to evaluate and iteratively refine structured outputs. In Precup, Doina and Teh, Yee Whye (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1341–1351, International Convention Centre, Sydney, Australia, 2017. PMLR.

Johnson, Justin, Krishna, Ranjay, Stark, Michael, Li, Li-Jia, Shamma, David A., Bernstein, Michael S., and Li, Fei-Fei. Image retrieval using scene graphs. In *Proc. Conf. Comput. Vision Pattern Recognition*, pp. 3668–3678, 2015.

Johnson, Justin, Gupta, Agrim, and Fei-Fei, Li. Image generation from scene graphs. *arXiv preprint arXiv:1804.01622*, 2018.

Khalil, Elias, Dai, Hanjun, Zhang, Yuyu, Dilkina, Bistra, and Song, Le. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6351–6361, 2017.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv: 1412.6980*, abs/1412.6980, 2014.

Krishna, Ranjay, Zhu, Yuke, Groth, Oliver, Johnson, Justin, Hata, Kenji, Kravitz, Joshua, Chen, Stephanie, Kalantidis, Yannis, Li, Li-Jia, Shamma, David A, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 282–289, 2001.

Liao, Wentong, Yang, Michael Ying, Ackermann, Hanno, and Rosenhahn, Bodo. On support relations and semantic scene graphs. arXiv preprint arXiv:1609.05834, 2016.

Lin, Guosheng, Shen, Chunhua, Reid, Ian, and van den Hengel, Anton. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, pp. 361–369, 2015.

Lu, Cewu, Krishna, Ranjay, Bernstein, Michael S., and Li, Fei-Fei. Visual relationship detection with language priors. In *European Conf. Comput. Vision*, pp. 852–869, 2016.

Meshi, O., Sontag, D., Jaakkola, T., and Globerson, A. Learning efficiently with approximate inference via dual losses. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 783–790, New York, NY, USA, 2010. ACM.

Newell, Alejandro and Deng, Jia. Pixels to graphs by associative embedding. In *Advances in Neural Information Processing Systems 30 (to appear)*, pp. 1172–1180. Curran Associates, Inc., 2017.

Newell, Alejandro, Huang, Zhiao, and Deng, Jia. Associative embedding: End-to-end learning for joint detection and grouping. In *Neural Inform. Process. Syst.*, pp. 2274–2284. Curran Associates, Inc., 2017.

Pei, Wenzhe, Ge, Tao, and Chang, Baobao. An effective neural network model for graph-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computationa Linguistics*, pp. 313–322, 2015.

Plummer, Bryan A., Mallya, Arun, Cervantes, Christopher M., Hockenmaier, Julia, and Lazebnik, Svetlana. Phrase localization and visual relationship detection with comprehensive image-language cues. In *ICCV*, pp. 1946–1955, 2017.

Raposo, David, Santoro, Adam, Barrett, David, Pascanu, Razvan, Lillicrap, Timothy, and Battaglia, Peter. Discovering objects and their relations from entangled scene representations. arXiv preprint arXiv:1702.05068, 2017.

Schwing, Alexander G and Urtasun, Raquel. Fully connected deep structured networks. *ArXiv e-prints*, 2015.

Shelhamer, Evan, Long, Jonathan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. *Proc. Conf. Comput. Vision Pattern Recognition*, 39(4):640–651, 2017.

Taskar, B., Guestrin, C., and Koller, D. Max margin Markov networks. In Thrun, S., Saul, L., and Schölkopf, B. (eds.), *Advances in Neural Information Processing Systems 16*, pp. 25–32. MIT Press, Cambridge, MA, 2004.

Xu, Danfei, Zhu, Yuke, Choy, Christopher B., and Fei-Fei, Li. Scene Graph Generation by Iterative Message Passing. In *Proc. Conf. Comput. Vision Pattern Recognition*, pp. 3097–3106, 2017.

Zaheer, Manzil, Kottur, Satwik, Ravanbakhsh, Siamak, Poczos, Barnabas, Salakhutdinov, Ruslan R, and Smola, Alexander J. Deep sets. In *Advances in Neural Information Processing Systems 30*, pp. 3394–3404. Curran Associates, Inc., 2017.

Zellers, Rowan, Yatskar, Mark, Thomson, Sam, and Choi, Yejin. Neural motifs: Scene graph parsing with global context. *arXiv preprint arXiv:1711.06640*, abs/1711.06640, 2017.

Zheng, Shuai, Jayasumana, Sadeep, Romera-Paredes, Bernardino, Vineet, Vibhav, Su, Zhizhong, Du, Dalong, Huang, Chang, and Torr, Philip HS. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1529–1537, 2015.

7 Supplementary Material

This supplementary material includes: (1) Visual illustration of the proof of Theorem 1. (2) Explaining how to integrate an attention mechanism in our GPI framework. (3) Additional details about GPI architecture for scene graphs. (4) Additional evaluation method to further analyze and compare our work with baselines.

7.1 Theorem 1: Illustration of Proof

7.2 Characterizing Permutation Invariance: Attention

Attention is a powerful component which naturally can be introduced into our GPI model. We now show how attention can be introduced in our framework. Formally, we learn attention weights for the neighbors j of a node i, which scale the features $\boldsymbol{z}_{i,j}$ of that neighbor. We can also learn different attention weights for individual features of each neighbor in a similar way.

Let $w_{i,j} \in \mathbb{R}$ be an attention mask specifying the weight that node i gives to node j:

$$w_{i,j}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j) = e^{\beta(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j)} / \sum_t e^{\beta(\boldsymbol{z}_i, \boldsymbol{z}_{i,t}, \boldsymbol{z}_t)} \tag{4}$$

where β can be any scalar-valued function of its arguments (e.g., a dot product of $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ as in standard attention models). To introduce attention we wish $\boldsymbol{\alpha} \in \mathbb{R}^e$ to have the form of weighting $w_{i,j}$ over neighboring feature vectors $\boldsymbol{z}_{i,j}$, namely, $\boldsymbol{\alpha} = \sum_{j \neq i} w_{i,j} \boldsymbol{z}_{i,j}$.

To achieve this form we extend $\boldsymbol{\phi}$ by a single entry, defining $\boldsymbol{\phi} \in \mathbb{R}^{e+1}$ (namely we set $L = e + 1$) as $\boldsymbol{\phi}_{1:e}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j) = e^{\beta(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j)} \boldsymbol{z}_{i,j}$ (here $\boldsymbol{\phi}_{1:e}$ are the first e elements of $\boldsymbol{\phi}$) and $\boldsymbol{\phi}_{e+1}(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}; \boldsymbol{z}_j) = e^{\beta(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j)}$. We keep the definition of $\boldsymbol{s}_i = \sum_{j \neq i} \boldsymbol{\phi}(\boldsymbol{z}_i, z_{i,j}, \boldsymbol{z}_j)$. Next,

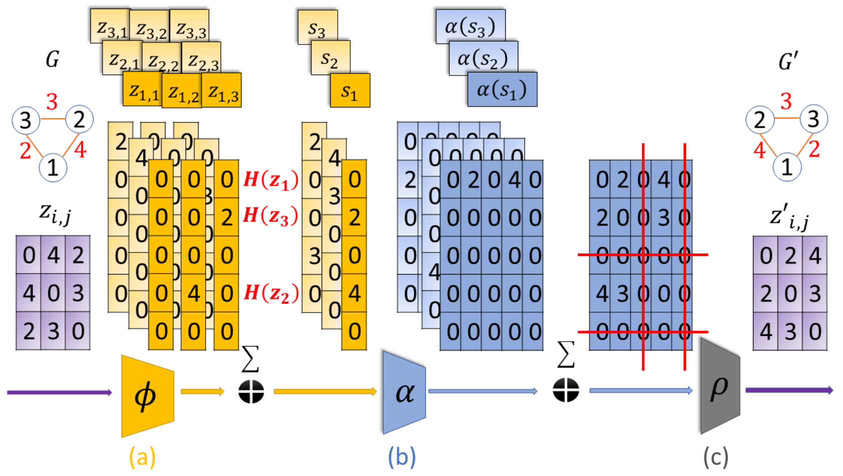


Figure 5: Illustration of the proof of Theorem 1 using a specific construction example. Here H is a hash function of size $L = 5$ such that $H(1) = 1, H(3) = 2, H(2) = 4$, G is a three-node input graph, and $\boldsymbol{z}_{i,j} \in \mathbb{R}$ are the pairwise features (in purple) of G. **(a)** ϕ is applied to each $\boldsymbol{z}_{i,j}$. Each application yields a vector in $\mathbb{R}^5$. The three dark yellow columns correspond to $\phi(\boldsymbol{z}_{1,1})$, $\phi(\boldsymbol{z}_{1,2})$ and $\phi(\boldsymbol{z}_{1,3})$. Then, all vectors $\phi(\boldsymbol{z}_{i,j})$ are summed over j to obtain three $\boldsymbol{s}_i$ vectors. **(b)** $\boldsymbol{\alpha}$'s (blue matrices) are an outer product between $\mathbb{1}\left[H(\boldsymbol{z}_i)\right]$ and $\boldsymbol{s}_i$ resulting in a matrix of zeros except one row. The dark blue matrix corresponds for $\alpha(\boldsymbol{z}_1, \boldsymbol{s}_1)$. **(c)** All α's are summed to a 5×5 matrix, isomorphic to the original $\boldsymbol{z}_{i,j}$ matrix.

we define $\boldsymbol{\alpha} = \frac{\boldsymbol{s}_{i,1:e}}{\boldsymbol{s}_{i,e+1}}$ and substitute $\boldsymbol{s}_i$ and $\boldsymbol{\phi}$ to obtain the desired form as attention weights $w_{i,j}$ over neighboring feature vectors $\boldsymbol{z}_{i,j}$:

$$\boldsymbol{\alpha}(\boldsymbol{z}_i, \boldsymbol{s}_i) = \frac{\boldsymbol{s}_{i,1:e}}{\boldsymbol{s}_{i,e+1}} = \sum_{j \neq i} \frac{e^{\beta(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j)} \boldsymbol{z}_{i,j}}{\sum_{j \neq i} e^{\beta(\boldsymbol{z}_i, \boldsymbol{z}_{i,j}, \boldsymbol{z}_j)}} = \sum_{j \neq i} w_{i,j} \boldsymbol{z}_{i,j}$$

A similar approach can be applied over $\boldsymbol{\alpha}$ and ρ to model attention over the outputs of $\boldsymbol{\alpha}$ as well (graph nodes).

7.3 Details of the Scene Graph Prediction Model

We further describe our Scene Graph Prediction (SGP) model described in section 5.2: "Scene-Graph Classification". Recall that the model is composed of three sub-networks ϕ, α, ρ and receives three types of inputs: entity features $\boldsymbol{z}_i$, relation features $\boldsymbol{z}_j$ and entity-entity pairwise features $\boldsymbol{z}_{i,j}$ which is the relation features between entity i and entity j.

The entity features include: (1) entity confidence learned by the baseline model. (2) bounding box given as a `(x, y, width, height)`. (3) number of entities with smaller/bigger size. (4) number of entities to the left/right/above/below. (5) number of entities with higher/lower confidence. (6) For Linguistic model only: word embedding of the most probable class. Word vectors were learned with GLOVE from the ground-truth captions of Visual Genome.

The relation features include: (1) relation confidence learned by the baseline model. (2) For the Linguistic model only: word embedding of the most probable class.

The ϕ network is implemented as a single fully-connected (FC) layer that outputs a vector of size 500. The α network is implemented as single FC layer that outputs a vector of size 500. The ρ network is implemented as FC network with 3 hidden layers of size 500 and outputs for entity vector of size 150 (the entity probabilities vector) and relation vector of size 51 (the relation probabilities vector). For the attention mechanism we further implement a network similar to ϕ and α that receives the same inputs, but outputs scores used for the attention mechanism.

7.4 Scene Graph Results

In the main paper, we described the results for the two prediction tasks: SGCls and PredCls, as defined in section 5.2.1: "Experimental Setup and Results". To further analyze our module, we compare the best variant, GPI: LINGUISTIC, per relation to two baselines: (Lu et al., 2016) and Xu et al. (2017). Table 2, specifies the PredCls recall@5 of the 20-top frequent relation classes. The GPI module performs better in almost all the relations classes.

RELATION	(LU ET AL., 2016)	(XU ET AL., 2017)	LINGUISTIC
ON	**99.71**	99.25	99.3
HAS	98.03	97.25	**98.7**
IN	80.38	88.30	**95.9**
OF	82.47	96.75	**98.1**
WEARING	98.47	98.23	**99.6**
NEAR	85.16	**96.81**	95.4
WITH	31.85	88.10	**94.2**
ABOVE	49.19	79.73	**83.9**
HOLDING	61.50	80.67	**95.5**
BEHIND	79.35	**92.32**	91.2
UNDER	28.64	52.73	**83.2**
SITTING ON	31.74	50.17	**90.4**
IN FRONT OF	26.09	59.63	**74.9**
ATTACHED TO	8.45	29.58	**77.4**
AT	54.08	70.41	**80.9**
HANGING FROM	0.0	0.0	**74.1**
OVER	9.26	0.0	**62.4**
FOR	12.20	31.71	**45.1**
RIDING	72.43	89.72	**96.1**

Table 2: Recall@5 of PredCls for the 20-top relations ranked by their frequency, as in (Xu et al., 2017)