eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Computer Science Degree

Computation

End of Degree Project

Inferring pictures layout from given text descriptions

Author

Carlos Domínguez Becerril

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Computer Science Degree

Computation

End of Degree Project

Inferring pictures layout from given text descriptions

Author

Carlos Domínguez Becerril

Instructors

Gorka Azkune Galparsoro and Oier López de Lacalle Lecuona

Abstract

Generating an image from a textual description is a complex task when the number of objects and relations specified on it is high, since the number of different possible solutions is very large. In this regard, humans tend to first imagine the layout of the described scene before painting it. We name this first step text-to-layout and we explore it in depth in this project.

This project explores various approaches that allow to infer the spatial layout of a picture given the text that describes it. Natural Language Processing (NLP) techniques are applied to represent and use the most important words and expressions in the text to obtain a coherent layout. More precisely, this project explores if a structured version of the text based on graphs can contribute to better represent the relations between objects.

Two different types of architectures have been developed based on graph convolutional neural networks (GCNN). The first one consists in obtaining directly the bounding boxes of the objects from the nodes of the graph, using hand-engineered heuristics. The second one uses a sequence-to-sequence (seq2seq) architecture with the same GCNN-based encoder as the first approach, but adding a decoder that learns to generate the layout sequentially. Furthermore, to evaluate the quality of the generated layouts new metrics are proposed, since the metrics currently used do not cover all the complexity associated with this task.

As the result of this exploration, we found systems that outperform the previous state-of-the-art in all the metrics on the MSCOCO dataset. Moreover, a visual comparison of the generated layouts shows a better spatial relationship between the objects in the scene. A thorough comparison of the different systems is provided through the analysis of the newly proposed metrics and the visual quality of the layouts.

Contents

List of figures

List of tables

CHAPTER 1

Introduction

Generating realistic images that match given text descriptions is a challenging problem and has tremendous potential in different applications, such as image editing, video games, and computer-aided design. Recently, thanks to the success of generative adversarial networks (GANs) [Goodfellow et al., 2014] in generating realistic images, text-to-image generation has made remarkable progress. Nowadays, GANs can generate realistic images conditioned on given text descriptions [Li et al., 2019a], nevertheless, other architectures such as multimodal transformers also offer good results [Ramesh et al., 2021].

Continuous improvements and innovations in these technologies have resulted in increasing the realism of the pictures that work especially well in simple scenes where one main object stands out (animals, faces, flowers, and so on) [Reed et al., 2016] [Zhang et al., 2017] [Zhu et al., 2019]. Consequently, there is a need to develop and refine new technologies for image synthesis of complex scenes composed of various objects and places, as well as their relationships.

In this regard, several works in the field of image generation from text follow a three-step process [Hong et al., 2018] [Justin et al., 2018] [Li et al., 2019b]:

- Generate the spatial composition of the objects contained in the text (i.e. size and position), taking into account the relationships described in the text. For example, in Figure 1.1 if "a person is talking on the phone while walking a dog", we can obtain different valid layouts (such as *a*, *b*, and *c*). In this case, the spatial layout should include a person bounding box which is taller than wider, a phone bounding box

located inside the person bounding box, specifically in the top part, which must be smaller than the person, and a dog that should be close to the bottom of the person with a bounding box taller than wider (examples *a* and *b*) or vice versa (example *c*). These are only three possible layouts of an infinite number of them.

- Generate segmentation maps of the objects generated in step 1. These segmentation maps will add shape to the bounding boxes which are going to be close to the final output.
- Generate the final image using the segmentation maps from step 2 and the original text.

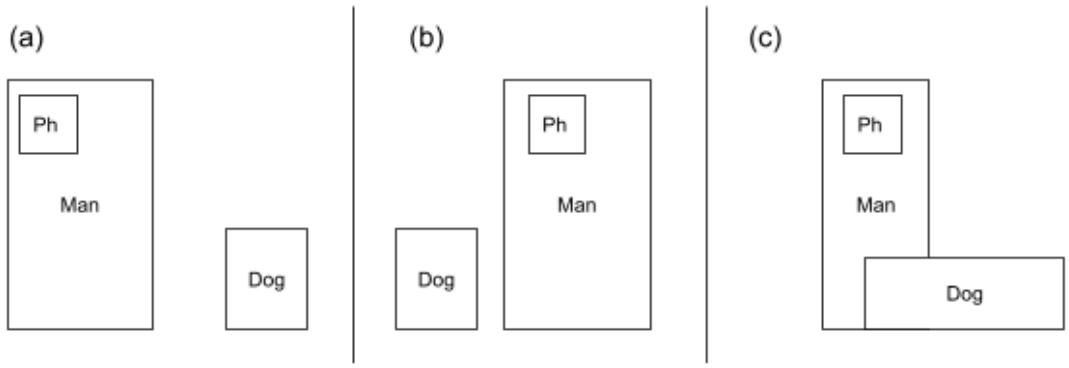


Figure 1.1: Example of different valid layouts for the description "a person is talking on the phone while walking a dog".

This process mimics to some extent the way humans draw, by first composing a scene and then painting the details. Despite the importance of the first step, composing the scene coherently with text, few researchers have devoted themselves to develop and evaluate it thoroughly. We think that the quality and reality of complex scenes are heavily attached to the layout and therefore, a deep research in the task of text-to-layout must be done. Moreover, this task shows the capacity of current systems to learn and extract spatial knowledge from text descriptions that usually are omitted but implied in the text. For example, given the sentence "woman riding a horse", the description does not say that the woman is on top of the horse but the verb "riding" with nouns "woman" and "horse" implies exactly that. Nevertheless, the difficulty of this task relies also on the human perception, which is different for every individual and layouts that seem correct for someone, may be incorrect for others and vice versa. Therefore, it is important to explore existing models, propose improvements and evaluate their performance when composing the scenes. This project

will not only be devoted to devising new models but also to proposing new forms of automatic evaluation that better measure the spatial composition of scenes, since the metrics currently used do not cover all the complexity associated with this task.

The contributions of this project consists of two systems that use a structured version of the text based on graphs and processed by a GCNN. The first one, called SG2BB that obtains directly the bounding boxes of the objects from the nodes of the graph, using hand-engineered heuristics. The second one, called GCN2LY that uses a seq2seq architecture with the same GCNN-based encoder as the first approach, but adding a decoder to learn to generate the layout sequentially. Moreover, this project also contributes to developing new metrics in order to better measure the quality of the generated layouts regarding the spatial composition of the scene and sizes of the objects drawn. The code developed to implement the architectures and perform the experiments can be found in the GitHub repository.

This project will explore the state-of-the-art systems in computer vision, natural language processing, and deep learning for the improvement of these, which have immediate practical applications in various fields related to image creation and editing.

CHAPTER 2

Deep learning for text-to-layout

To understand the following chapters well, some background knowledge is introduced that will become essential in the coming explanations. Because it is not possible to go in-depth into all the theoretical aspects that would be required to fully understand the content in the next chapters, the reader is assumed to have an introductory level understanding of basic Deep Learning (DL) techniques. The reader is referred to [Goodfellow et al., 2016] and [Wu et al., 2019b] for a much more complete coverage on deep learning techniques.

Text-to-layout systems are usually presented as the first step of text-to-image systems in the literature [Justin et al., 2018] [Li et al., 2019b]. In consequence, researchers have not paid too much attention to the development and evaluation of this task, but several solutions can be found based on different neural network architectures such as multilayer perceptrons (MLP), recurrent neural networks (RNN), graph convolutional neural networks (GCNN), and transformers.

2.1 Neural network architectures

2.1.1 Multilayer perceptron

The goal of a multilayer perceptron is to approximate some function f^*. For example, a classifier $y = f^*(x)$ maps an input x to a category y. A multilayer perceptron defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation.

Multilayer perceptrons are called multilayer because they are typically represented by composing together many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together. For example, given three functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain, to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. These chained structures are the most commonly used structures of neural networks, where $f^{(1)}$ refers to the first layer, $f^{(2)}$ to the second layer, and $f^{(3)}$ to the third layer. The first layer of a multilayer perceptron is called the input layer, the intermediate layers are called hidden layers, and the final layer is called the output layer. The number of layers determines the depth of the model. An example of this architecture can be seen in Figure 2.1.

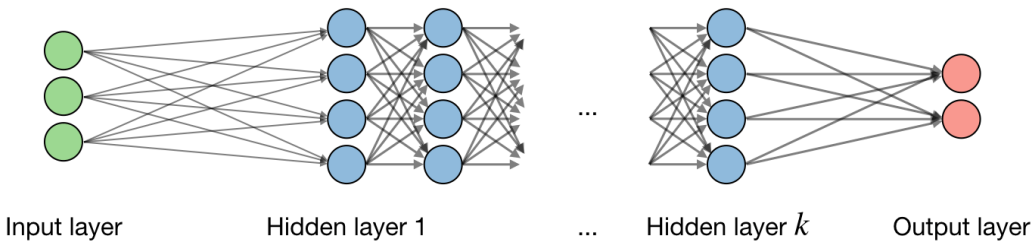


Figure 2.1: MLP architecture diagram with an input layer, k hidden layers, and an output layer. Source: Stanford university.

If a multilayer perceptron has a linear activation function in all neurons, that is, a linear function that maps the weighted inputs to the output of each neuron, then linear algebra shows that any number of layers can be reduced to a two-layer input-output model. In MLPs some neurons use a nonlinear activation function that was developed to model the frequency of action potentials, or firing, of biological neurons.

Two common non-linear activation functions are the sigmoid (σ) and the rectified linear unit (*ReLU*) functions described by Equations 2.1 and 2.2:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

$$ReLU(x) = max(0,x) \tag{2.2}$$

There are various ways to learn the suitable weights of a neural network but this project will only focus on backpropagation [Rumelhart et al., 1986] combined with gradient-based optimization techniques such as stochastic gradient descent [Robbins, 2007].

2.1.2 Recurrent neural networks

Recurrent neural networks (RNNs) are a family of deep learning architectures that are specialized in processing data of sequential nature. RNNs are well-suited for several NLP tasks due to the sequential essence of language and they are used extensively in the field. Some examples of the use of RNNs in NLP and language-related topics are machine translation [Wu et al., 2016], language modeling [Józefowicz et al., 2016], reading comprehension [Shen et al., 2017], question-answering [Andreas et al., 2016], and text summarization [Mahmood and Len, 2017] to name a few.

There are many types of RNNs, however, they are all based on the same fundamental ideas. A sequence of elements are processed one by one and to process any element, two inputs are needed: a vector representation of the element and a state vector (also called hidden-state) which encodes all the elements seen so far. Using these two inputs a RNN cell will produce the next hidden-state as an output which also can be used to feed the next RNN cell state.

A RNN at time-step t has hidden-state h_{t-1} and processes an element x_t from a sequence x. The RNN computes the next hidden-state h_t as function f of the previous hidden-state h_{t-1} and the element that is being processed x_t as seen in Figure 2.2.

f needs to be suitable for DL techniques, therefore, it has to be a parametric function $f = f_\theta$ with parameters θ and differentiable with respect to θ, i.e, $\exists \frac{df}{d\theta}$ to be able to use gradient-based methods and optimize the parameters efficiently.

$$h_t = f(x_t, h_{t-1}) \tag{2.3}$$

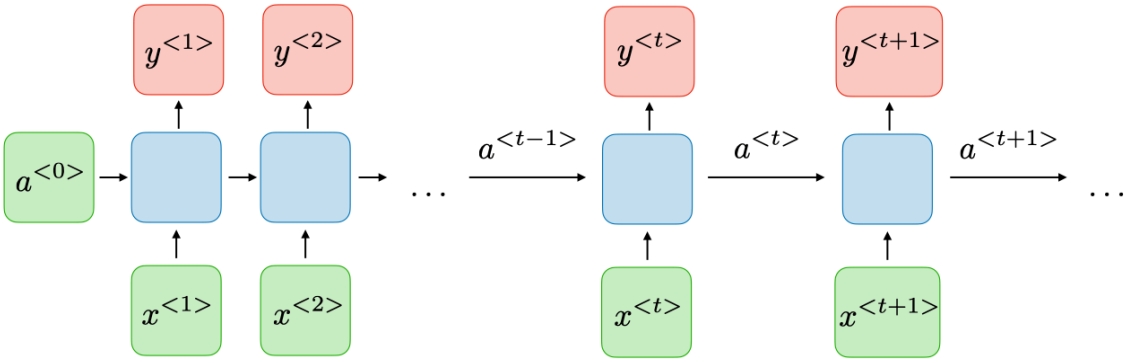
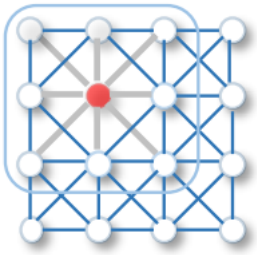


Figure 2.2: RNN architecture diagram, where a is the hidden-state. Source: Stanford university.

Finally, f should be a non-linear function so that the capacity of the model can scale with the depth of the deep learning model. The choice of f is important, for that reason, different types of RNN architectures have been proposed such as Long-Short Term Memory cells (LSTM) [Hochreiter and Schmidhuber, 1997] or Gated Recurrent Units (GRU) [Cho et al., 2014] to name a few.

2.1.3 Graph convolutional neural networks

Graph neural networks were initially introduced by [Gori et al., 2005] and further elaborated by [Scarselli et al., 2009]. These early studies were related to recurrent graph neural networks (RGNN), where the target node's representation is learnt by iteratively propagating neighbour information until a stable fixed point is reached. However, this process is computationally expensive and recently there have been increasing efforts to overcome these challenges. The success of convolutional neural networks (CNNs) in computer vision (images can be considered as a special case of graphs as seen in Figure 2.3), encouraged the development of different types of methods that re-define the notion of convolution for graph data. This methods are under the umbrella of graph convolutional neural networks and are divided into two different approaches: the spectral-based approaches and the spatial-based approaches.



(a) 2D Convolution: Each pixel in an image is a node and the neighbours are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbours. The neighbours of a node are ordered and have a fixed size.

(b) Graph Convolution: The hidden representation of the red node can be obtained by averaging the value of the node features along with its neighbours. The neighbours of a node are unordered and variable in size.

Figure 2.3: 2D Convolution vs. Graph Convolution. Source: [Wu et al., 2019b].

To understand what a graph convolutional neural network is, first a minimal set of definitions are needed:

- **Graph:** A graph is represented as $G = (V, E)$ where V is the set of vertices, also called nodes and E is the set of edges. Let $v_i \in V$ denote a node and $e_{ij} = (v_i, v_j) \in E$ an edge pointing from v_j to v_i. The neighbourhood of a node v is defined as $N(v) = u \in V | (v,u) \in E$. The adjacency matrix A is an $n \times n$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ if $e_{ij} \notin E$. A graph may have node attributes X where $X \in \mathbb{R}^{n \times d}$ is a node feature matrix with $x_v \in \mathbb{R}^d$ representing the feature vector of a node v. At the same time, a graph may have edge attributes X^e , where $X^e \in \mathbb{R}^{m \times c}$ is an edge feature matrix with $x^e_{v,u} \in \mathbb{R}^c$ representing the feature vector of an edge (v,u).
- **Directed graph:** A directed graph is a graph with all edges directed from one node to another. An undirected graph is considered as a special case of directed graphs where there is a pair of edges with inverse directions if two nodes are connected. A graph is undirected if and only if the adjacency matrix is symmetric.
- **Spatial-Temporal Graph:** A spatial-temporal graph is an attributed graph where the node attributes change dynamically over time. The spatial-temporal graph is defined as $G^{(t)} = (V, E, X^{(t)})$ with $X^{(t)} \in \mathbb{R}^{n \times d}$.

GCNN generalize the operation of convolution from grid data to graph data. To generate a node v's representation, its own features x_v and neighbours' features x_u, where $u \in N(v)$, are aggregated. GCNN stack multiple graph convolutional layers to extract high-level node representations. Figure 2.4 shows a GCNN for node classification.

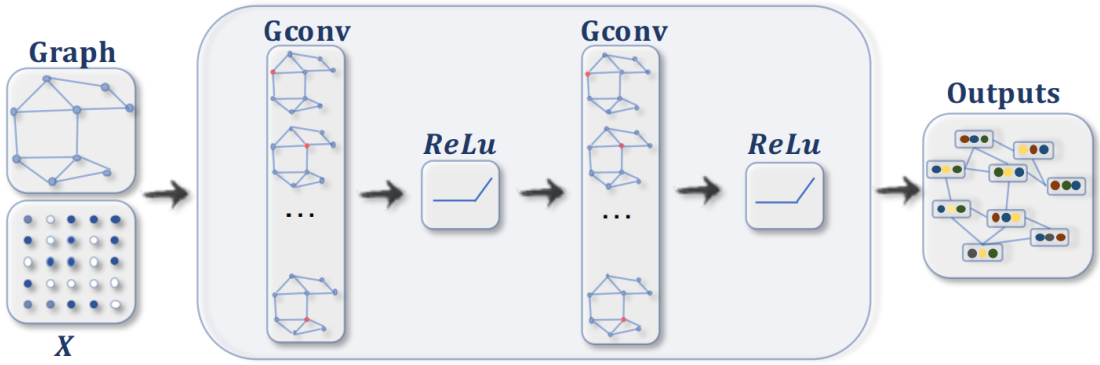


Figure 2.4: A CGNN with multiple graph convolutional layers. Each node's hidden representation is obtained by aggregating feature information from its neighbours. After feature aggregation, a non-linear transformation is applied to the resulted outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighbourhood. Source: [Wu et al., 2019b].

2.2 Seq2Seq architecture

Seq2seq [Sutskever et al., 2014] is a common DL technique used in NLP (and also in other fields) when dealing with tasks that require transforming data of sequential nature (e.g. a sentence) between representations.

A seq2seq architecture transforms a sequence into another sequence. It does so by employing two connected RNNs: a combination of an encoder and a decoder as seen in Figure 2.5. A key aspect of seq2seq methods is that the parameters of the whole architecture are learnt end-to-end, which means that the gradient of the loss of the decoder, is propagated through the whole computation graph that defines the architecture; even the encoder and the embedding tables at the very beginning of the graph.

Seq2seq architectures have been employed with success in many tasks that involve transforming a natural language fragment into another natural language fragment. Some examples are machine translation [Wu et al., 2016], conversational tasks [Andreas et al., 2016], and text summarization [Mahmood and Len, 2017].

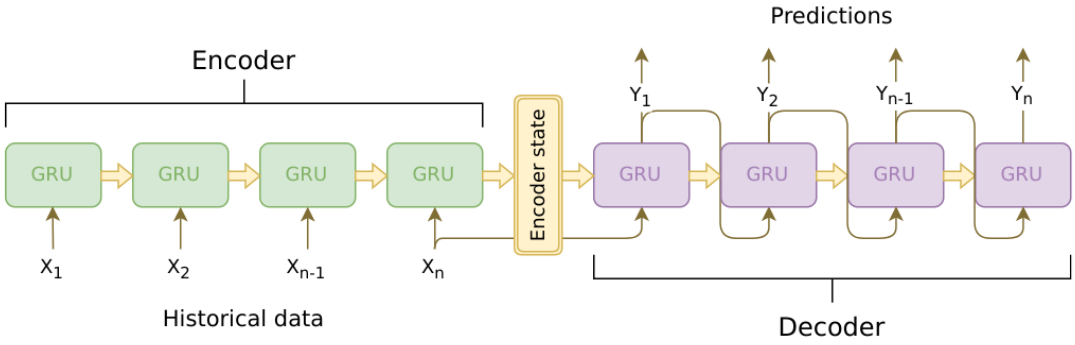


Figure 2.5: Diagram of a seq2seq model used for transforming historical data into a prediction. Source: Soohwan Kim's GitHub repository.

2.3 Word embeddings

In natural language processing (NLP), Word Embeddings is a term used for real-valued vectors that encode the meaning of a word such that the words that are closer in the vector space should be similar. This characteristic can be seen in Figure 2.6. Some examples of word embeddings are word2vec [Mikolov et al., 2013a], GloVe [Pennington et al., 2014], fastText [Mikolov et al., 2013b], and Flair embeddings [Akbik et al., 2019] to name a few.

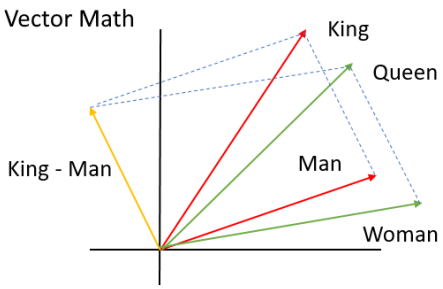


Figure 2.6: Words similarity. Similar words should have vectors that are close to each other. Source: Chicago University.

In this project, we make use of GloVe embeddings. Global vectors for word representation (GloVe) is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

2.4 Text-to-layout architectures

Text-to-layout is the task of generating the layout of a picture given the text that describes it. State-of-the-art methods for generating images from sentences struggle to represent complex sentences with many objects. In this project two different approaches of text-to-image (generation of pictures from text) have been tested: Scene Graph to Image (SG2IM) [Justin et al., 2018] and Object-GAN (Obj-GAN) [Li et al., 2019b]. In both approaches, the layout of the picture is first generated before drawing the final scene.

Text-to-layout defines the concept of bounding box as $b \in \mathbb{R}^4$ where the first two elements b_0 and b_1 refers to the upper-left corner or center and b_2 and b_3 to the lower-right corner or the width and height of the bounding box. Objects are defined as a set of given object categories C.

2.4.1 SG2IM

Image generation from scene graphs (SG2IM) uses graph convolutions to first process input scene graphs, then compute the scene layout by predicting bounding boxes and segmentation masks for objects, and finally, convert the layout to an image using a cascaded refinement network.

The architecture of this approach can be seen in Figure 2.7.

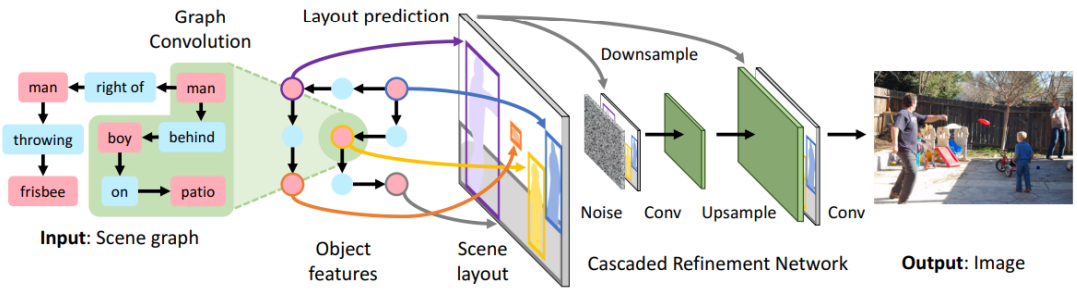


Figure 2.7: Overview of the SG2IM architecture. Source: [Justin et al., 2018].

SG2IM system can be split into two parts: layout generation and image generation. This project gives more importance to the layout generation which SG2IM does in three steps:

1. First the scene graphs are defined. Scene graphs describe objects and relationships given a set of object categories C and a set of relationship categories R, then a scene graph is a tuple (O,E) where $O = \{o_1,...,o_n\}$ is a set of objects with each $o_i \in C$, and $E \subseteq O \times R \times O$ is a set of directed edges of the form (o_i,r,o_j) where $o_i,o_j \in O$ and $r \in R$. For MSCOCO dataset [Lin et al., 2014], the graphs are generated in execution time taking into account all the ground truth objects and using six mutually exclusive geometric relationships: *left of, right of, above, below, inside* and *surrounding*. Worth mentioning that text is not involved here, therefore, the objects that appear in the graph are not directly related to the text description of the scene, which allows to create complex scenes.

2. After defining the scene graphs a graph convolutional neural network is used to process the scene graphs in an end-to-end manner where the output vectors are a function of a neighbourhood of their corresponding inputs so that each graph convolution layer propagates information along edges on the graph.

3. Finally, after processing the scene graphs a multilayer perceptron is used to generate the bounding boxes $b_i = (x_0,y_0,x_1,y_1)$ given the nodes embeddings.

For this project, the remaining steps that SG2IM uses for generating the picture are not relevant.

2.4.2 Obj-GAN

Object-driven Attentive Generative Adversarial Networks (Obj-GAN) allows text-to-image synthesis of complex scenes using a two-step (layout-image) process. First, the system takes a sentence as input and generates the layout, a sequence of objects specified by their bounding boxes, shapes, and class labels $B_t = (l_t,b_t)$ with $b_t = (x,y,w,h) \in \mathbb{R}^4$ and $l_t \in C$, where C is a set of given object categories. The second step uses a multistage generative network with two generators (G_0,G_1). The first generator G_0 generates a low-resolution image conditioned on the global sentence vector and the pre-generated layout, then the second generator G_1 refines the details of the picture by paying attention to the most important words, generating a higher resolution image.

The architecture of this approach can be seen in Figure 2.8. Obj-GAN is the state-of-the-art for text-to-layout task and it will be used as the reference system to compare results.

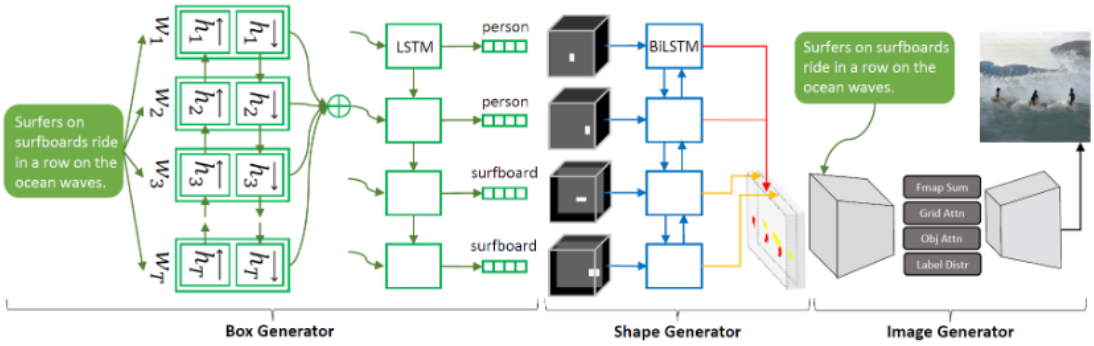


Figure 2.8: Overview of the Obj-GAN architecture. Source: [Li et al., 2019b].

Obj-GAN system can be split into two parts: layout generation and image generation. This project gives more importance to the layout generation which Obj-GAN does in two steps:

1. First, the text is encoded using a pre-trained bi-LSTM word vectors. This text encoder uses a Deep Attentional Multimodal Similarity Model (DAMSM) [Xu et al., 2017] with an attention mechanism which is able to compute the similarity between the generated image and the sentence using both, the global sentence information and the fine-grained word-level information.

2. Finally, the bounding boxes and object labels are generated sequentially using an LSTM with teacher forcing and attention conditioned on the global sentence vector.

For this project, the remaining steps that Obj-GAN uses for generating the picture are not relevant.

CHAPTER 3

MSCOCO Dataset

In this chapter, we introduce the dataset, the problems related with it, and the systems to generate the graphs, which includes information about the tools used, the structure to store all the information, and statistics related with the dataset.

The Microsoft Common Objects in COntext (MSCOCO) dataset [Lin et al., 2014] contains 91 common object categories, such as car, person, pizza, or zebra, with 80 of them having more than 5,000 labeled instances. In total the dataset has 2,500,000 labeled instances in 328,000 images. This dataset is widely used in computer vision, specifically in object detection and object segmentation, due to each picture containing 5 captions that describes the scene, object bounding boxes and labels that appear on it, and segmentation maps for each object.

Figure 3.1 shows a picture with one of the captions that describes the scene and its respective bounding boxes.

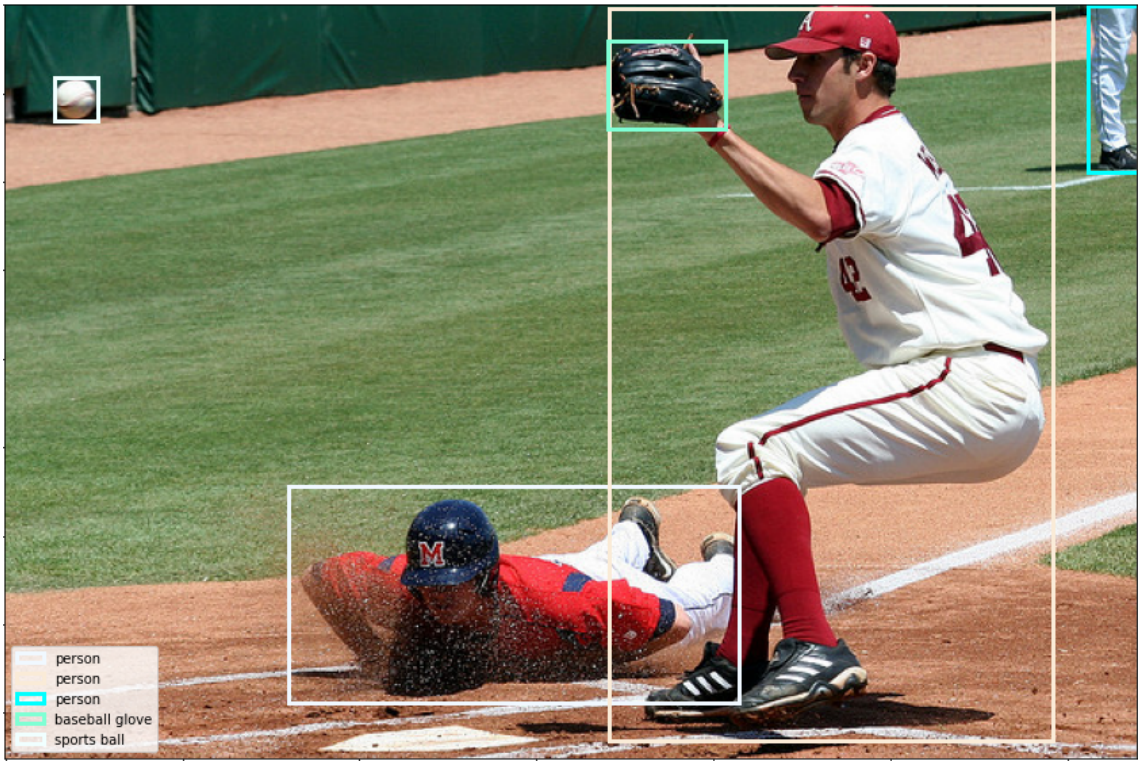


Figure 3.1: A caption that describes correctly the scene. Caption: A baseball player sliding into home plate, in a game. MSCOCO 2014.

3.1 Dataset related problems

In this project, MSCOCO 2014 is used as the reference dataset given its wide usage in the community. Nevertheless, using this dataset brings two main problems: the captions of some pictures do not describe well the scene and the number of bounding boxes of some pictures is high.

The first problem of the dataset is that it contains some captions that may not be suitable for the task due to the description of them being too specific for the pictures. An example of this problem can be seen in Figure 3.2. This type of captions are going to add noise to the task as there is not an easy way to remove them or reduce the impact on the system. In this project, we have decided not do anything to overcome this problem.

Figure 3.2: A caption that does not describe correctly the scene for being too specific. Caption: The intersection of East Canal Street North, near West 717. MSCOCO 2014.

The second problem of the dataset is that some pictures contain too many bounding boxes which adds complexity to the task due to the need to differentiate between the main and secondary objects (background objects). An example of this problem can be seen in Figure 3.3. In this regard, a human-based approach is used to solve this problem. Humans tend to draw first big objects, usually the most important ones in the scene, while leaving the smallest ones, usually background objects, for the end. Therefore, the dataset is pre-processed sorting the objects in descending order by their area size (from biggest area to smallest).

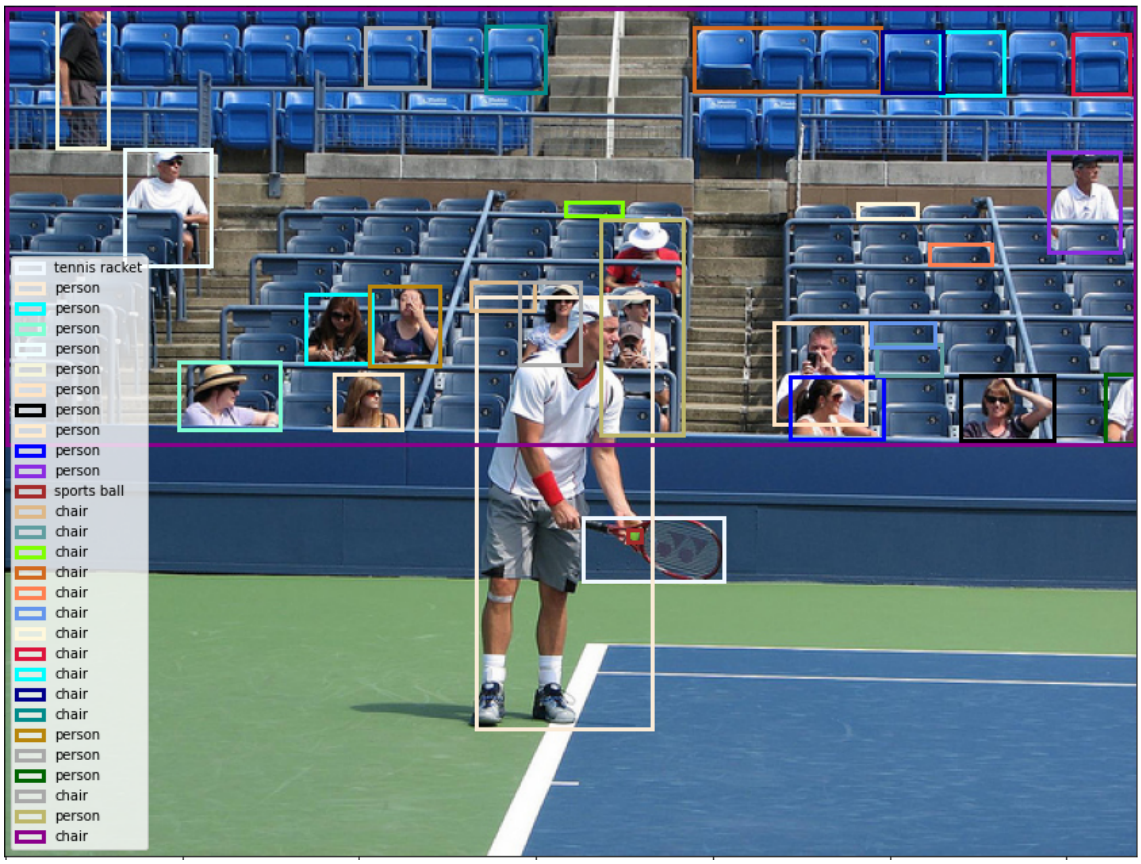


Figure 3.3: Picture with 30 bounding boxes. MSCOCO 2014.

3.2 Generation of graphs from text

Using graphs to represent the text ease the understanding of the objects and the relations between the objects that appear. We think that explicitly representing this information can contribute to representing better what is happening in the text, and therefore, improve text-to-layout systems.

Generating a structured version of the text using graphs is a challenging task due to the different types of representations available. In this project, we focus on two types of graphs: Abstract Meaning Representation (AMR) and Scene Graph Parser (SGP) graphs.

3.2.1 AMR

Abstract Meaning Representation (AMR) is a broad-coverage semantic formalism that encodes the meaning of a sentence as rooted, directed, and labeled graph, where nodes represent concepts and edges represent relations. AMR parsing is the task of transforming natural language text into AMR. The AMR graphs used contain 85 different relations. An example of this graph can be seen in Figure 3.4. The tool used to extract AMR graphs is the one referred in [Cai and Lam, 2020] paper and available in Github.

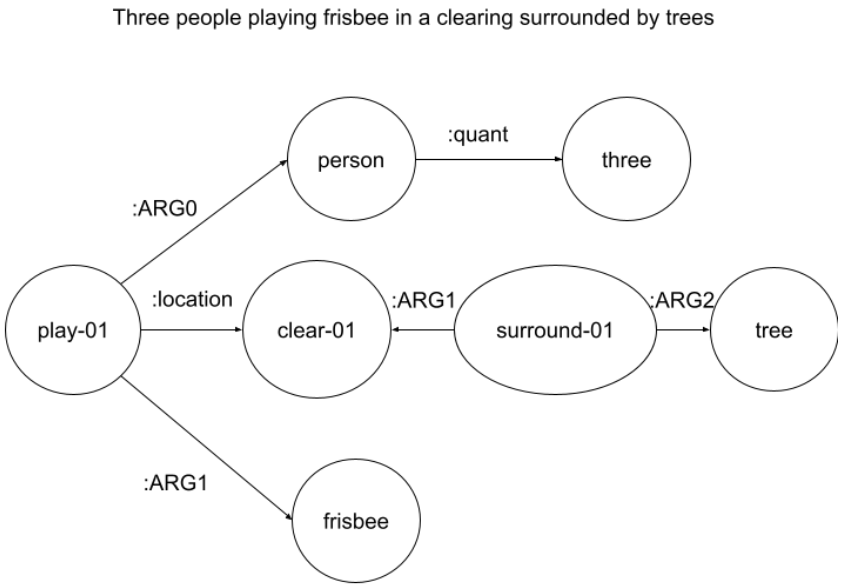


Figure 3.4: Example of an AMR graph.

Figure 3.4, shows an AMR graph. In this case, “play-01” has two frame arguments “person” and “frisbee” represented with “ARG0” and “ARG1” relations respectively. The text specifies that there are three persons, therefore, “person” has a quantity relation “:quant” that points to “three” node. Moreover, “play-01” action is done in a specific location represented with the relation “:location” that points to “clear-01” node. “surround-01” node has another two frame arguments “clear-01” and “tree” connected using “ARG0” and “ARG1” relations respectively.

AMR concepts (nodes) are either English words (“boy”), PropBank framesets (“want-01”) [Kingsbury and Palmer, 2002], or special keywords. Keywords include special entity types (“date-entity”, “world-region”, and so on.), quantities (“monetary-quantity”, “distance-quantity”, and so on), and logical conjunctions (“and”, and so on).

3.2.2 Scene graph parser

Scene Graph Parser is a python toolkit for parsing sentences (in natural language) into scene graphs (as symbolic representation) based on extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between “head” words and words, which modify those heads. An example of this type of graph can be found in Figure 3.5. The main difference between AMR and SGP graphs is that AMR has a set of predefined relations, while SGP uses the ones that appear in the text. The tool used to extract the SGP graphs is the one referred in [Wu et al., 2019a] paper and available in Github.

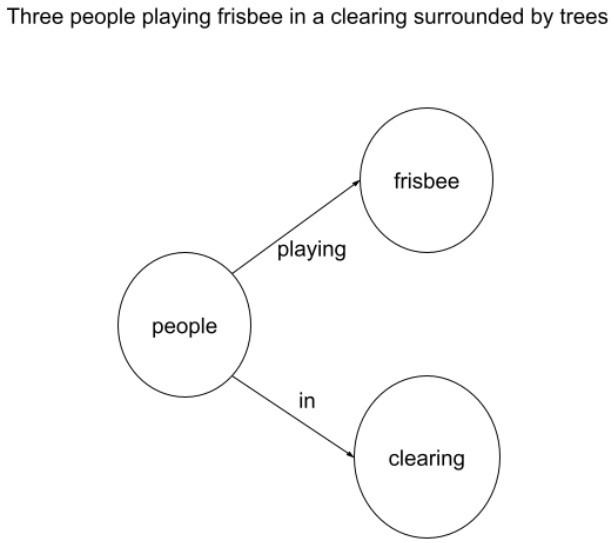


Figure 3.5: Example of an SGP graph.

Figure 3.5, shows a SGP graph. In this case, the relations and objects are extracted directly from the text. The node “people” is connected to “frisbee” with the relation “playing” as it is the action being performed. Moreover, the location where they are playing is specified with the relation “in” connected to “clearing” node.

3.3 Dataset structure

For the task of text-to-layout generation, this project builds on the MSCOCO dataset and further extends it with the basic information about the picture and either the graphs generated using AMR or SGP. The following structure is proposed to store the graphs:

```
{
    '00001':{
        "image_filename": "00001.jpg",
        "width": 256,
        "height": 256,
        "valid_captions": 2,
        "graphs":[
            {
                "caption_n": 1,
                "caption": "caption 1 ......",
                "objects": [["object1", 0], ["object2", 1], ["object3", 1]],
                "relations": ["relation1", "relation2"],
                "triples": [
                    [["object1", 0], "relation1", ["object2", 1]],
                    [["object2", 1], "relation2", ["object3", 2]]
                ]
            },
            {
                "caption_n": 2,
                "caption": "caption 2 ......",
                "objects": [["object1", 1], ["object2", 0]],
                "relations": ["relation1"],
                "triples": [
                    [["object1", 0], "relation1", ["object2", 1]]
                ]
            }
        ]
    },
    '00002'{
        ...
    }
}
```

For a given image ID each image contains the following information:

- Image filename: name of the file that contains the image.
- Width: width of the image in pixels.
- Height: height of the image in pixels.
- Valid captions: number of valid captions for a given image.
- Graphs: a graph for each caption with the following information:
  - Caption number: the identifier for a given caption.
  - Caption: text that describes the picture.
  - Objects: list of objects that appear in the graph. The first element is the name of the object and the second element specifies whether the object is valid or not (all objects are valid when processing the graphs using a GCNN. This can be used to remove some nodes afterwards).
  - Relations: list of relations that appear between objects in the graph.
  - Triples: list of triples with a subject–relation–object structure. Each subject/object contains an integer referencing the object in the list of objects that belongs to, which allows having more than one node with the same name.

3.4 The dataset in numbers

The following section shows the information that each dataset has using AMR and SGP graphs. The dataset is divided into three partitions: train, development, and testing containing 74504, 8279, and 40504 pictures respectively.

The following tables measure the mean, standard deviation, minimum, and maximum values regarding the number of valid captions that each partition has using both types of graphs. The training, development, and testing partitions information can be seen in Table 3.1.

This basic information shows that the number of valid captions for AMR and SGP graphs are similar in the three partitions, therefore, the dataset a priori seems balanced.

	Training		Development		Testing	
	AMR	SGP	AMR	SGP	AMR	SGP
mean	5.00	4.28	5.00	4.29	5.00	4.27
std	0.05	0.85	0.05	0.85	0.06	0.86
min	5.00	0.00	5.00	0.00	5.00	1.00
max	6.00	6.00	7.00	7.00	7.00	7.00

Table 3.1: Information about the number of valid captions that each picture has in training, development, and testing partitions. AMR stands for valid captions for AMR representation, whereas SGP denotes valid captions for SGP representation in all the tables.

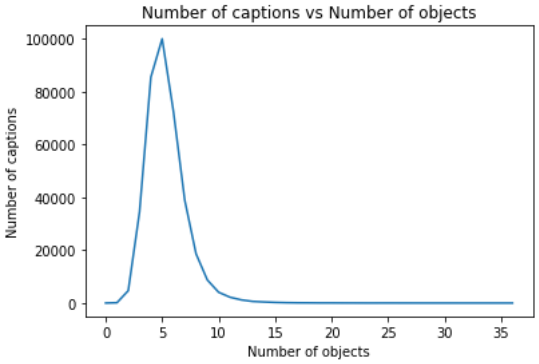
3.4.1 AMR

The following tables measure the mean, standard deviation, minimum, and maximum values regarding the number of objects, number of relations, and number of triples that each partition has for each AMR graph. To understand better the results graphical figures are included showing how these quantities are distributed among the number of captions. The training, development, and testing partitions information can be seen in Figures 3.6, 3.7, and 3.8 respectively.

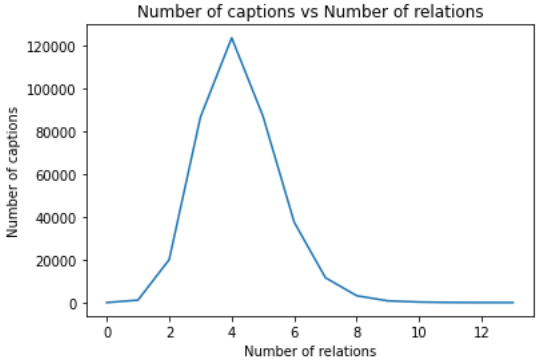
The tables show that the dataset has similar values for number of objects, number of relations, and number of triples per caption. Moreover, the figures show that most of the captions have between 3 and 10 objects, between 3 and 6 relations, and between 3 and 10 triples. Nevertheless, the data shows that some graphs contain no triples, no objects, or no relations, therefore, the dataset needs to be preprocessed to delete these graphs as they do not contribute any information.

	Number of objects	Number of relations	Number of triples
mean	6.36	4.23	5.54
std	1.81	1.25	1.93
min	0.00	0.00	0.00
max	40.00	13.00	39.00

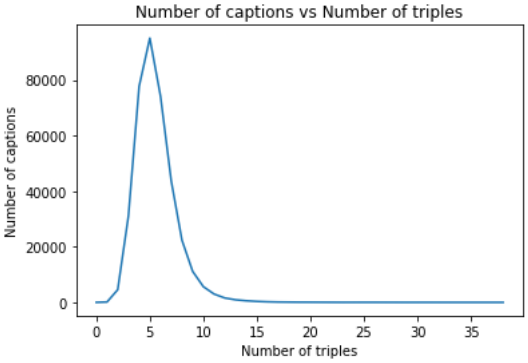
(a) Statistics for Number of objects, number of relations, and number of triples.



(b) Number of captions vs. Number of objects.



(c) Number of captions vs. Number of relations.

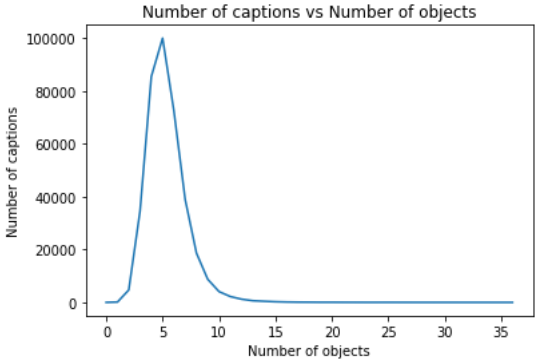


(d) Number of captions vs. Number of triples.

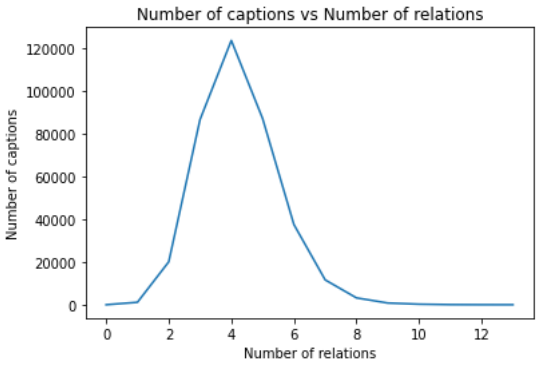
Figure 3.6: Table and figures showing the statistics for number of objects, number of relations, and number of triples that each caption has in the training partition.

	Number of objects	Number of relations	Number of triples
mean	6.37	4.25	5.56
std	1.82	1.26	1.94
min	2.00	1.00	1.00
max	33.00	15.00	34.00

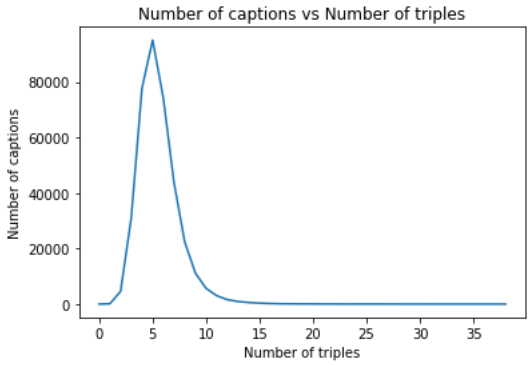
(a) Statistics for Number of objects, number of relations, and number of triples.



(b) Number of captions vs. Number of objects.



(c) Number of captions vs. Number of relations.

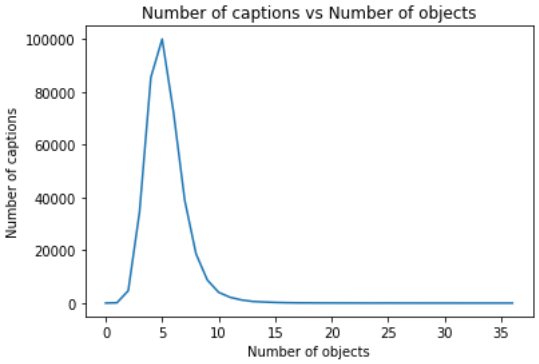


(d) Number of captions vs. Number of triples.

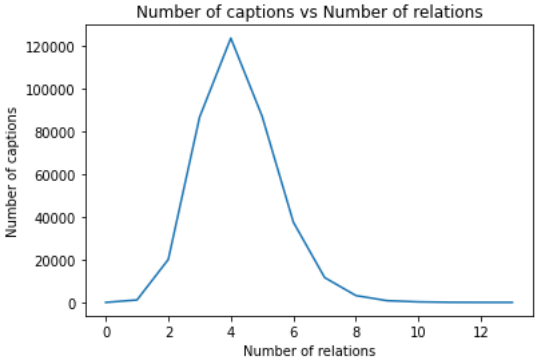
Figure 3.7: Table and figures showing the statistics for number of objects, number of relations, and number of triples that each caption has in the development partition.

	Number of objects	Number of relations	Number of triples
mean	6.34	4.22	5.53
std	1.79	1.25	1.91
min	0.00	0.00	0.00
max	34.00	14.00	33.00

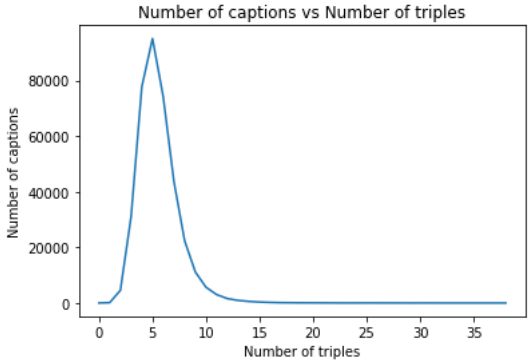
(a) Statistics for Number of objects, number of relations, and number of triples.



(b) Number of captions vs. Number of objects.



(c) Number of captions vs. Number of relations.



(d) Number of captions vs. Number of triples.

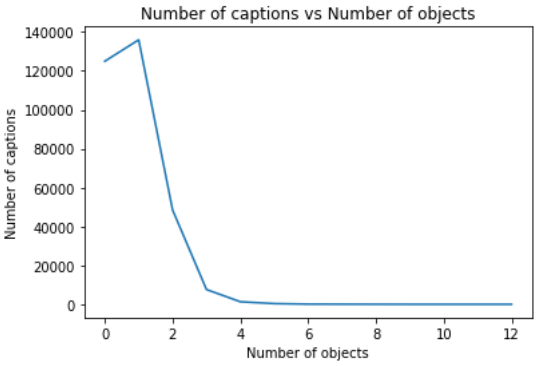
Figure 3.8: Table and figures showing the statistics for number of objects, number of relations, and number of triples that each caption has in the testing partition.

3.4.2 SGP

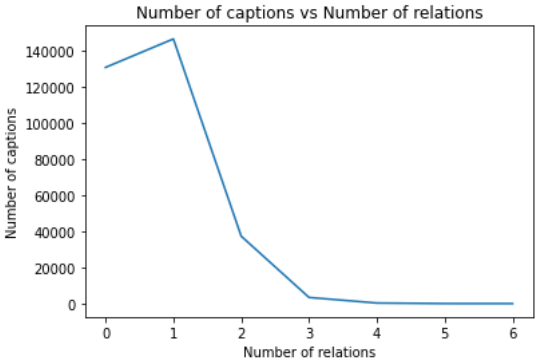
The following tables measure the mean, standard deviation, minimum, and maximum values regarding the number of objects, number of relations, and number of triples that each partition has for each SGP graph. To understand better the results graphical figures are included showing how these quantities are distributed among the number of captions. The training, development, and testing partitions information can be seen in Figures 3.9, 3.10, and 3.11 respectively.

	Number of objects	**Number of relations**	**Number of triples**
mean	2.83	1.73	1.79
std	0.83	0.71	0.77
min	2.00	1.00	1.00
max	14.00	7.00	11.00

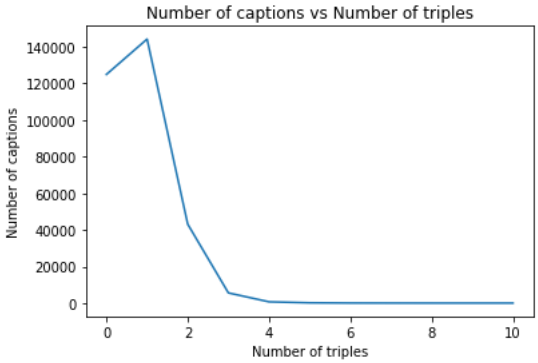
(a) Statistics for Number of objects, number of relations, and number of triples.



(b) Number of captions vs. Number of objects.



(c) Number of captions vs. Number of relations.

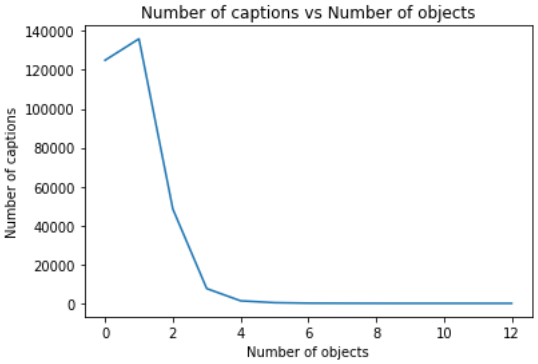


(d) Number of captions vs. Number of triples.

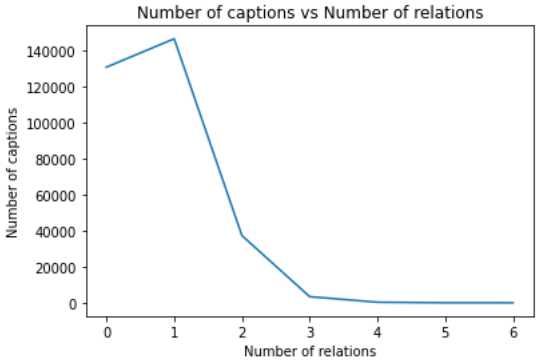
Figure 3.9: Table and figures showing the statistics for number of objects, number of relations, and number of triples that each graph has in the training partition.

	Number of objects	Number of relations	Number of triples
mean	2.84	1.74	1.80
std	0.83	0.72	0.77
min	2.00	1.00	1.00
max	14.00	7.00	11.00

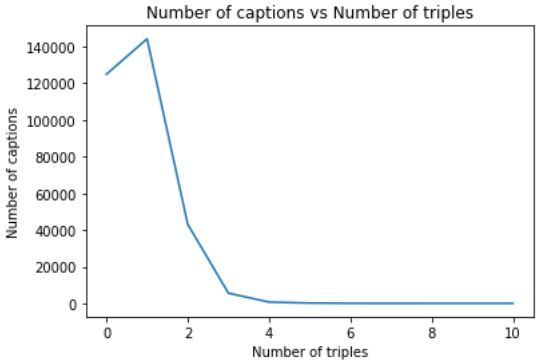
(a) Number of objects, number of relations, and number of triples statistics.



(b) Number of captions vs. Number of objects.



(c) Number of captions vs. Number of relations.

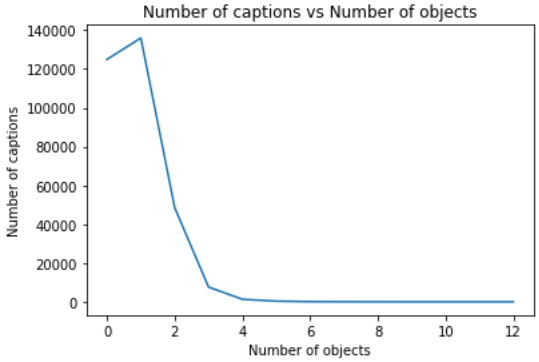


(d) Number of captions vs. Number of triples.

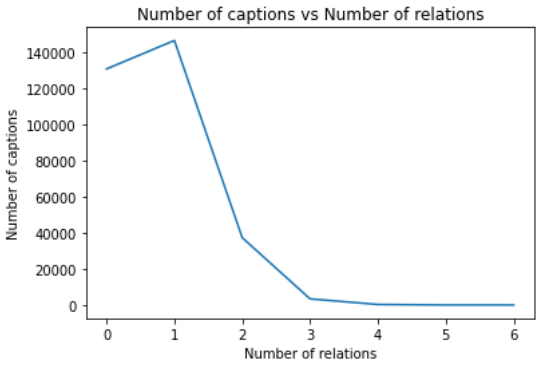
Figure 3.10: Table and figures showing the statistics for number of objects, number of relations, and number of triples that each graph has in the development partition.

	Number of objects	Number of relations	Number of triples
mean	2.83	1.73	1.79
std	0.82	0.71	0.76
min	2.00	1.00	1.00
max	12.00	7.00	9.00

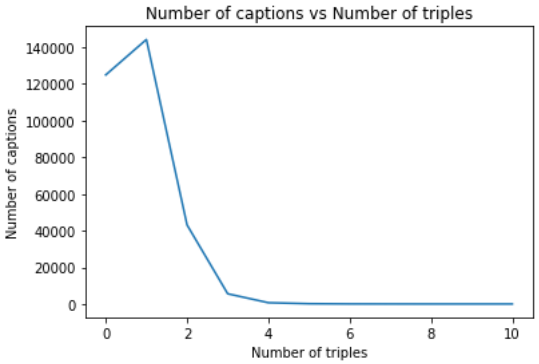
(a) Statistics for Number of objects, number of relations, and number of triples.



(b) Number of captions vs. Number of objects.



(c) Number of captions vs. Number of relations.



(d) Number of captions vs. Number of triples.

Figure 3.11: Table and figures showing the statistics for number of objects, number of relations, and number of triples that each graph has in the testing partition.

The tables show that the dataset has similar values for number of objects, number of relations, and number of triples per caption. Moreover, the figures show that most of the captions have 1 or 2 objects, 1 or 2 relations, and 1 or 2 triples. The data shows that some graphs contain no triples, no objects, or no relations, therefore, the dataset needs to be preprocessed to delete these graphs as they do not contribute any information.

SGP graphs show a lack of information compared to AMR graphs when taking into account the number of objects, number of relations, and number of triples. Creating a structured version of the text that contains enough information to represent it correctly is important, and we think that SGP graphs may lack essential information that AMR graphs have, therefore, this project will focus only on AMR graphs.

CHAPTER 4

Developed architectures

In this chapter, the two architectures developed will be presented. The first one consists in obtaining directly the bounding boxes of the objects from the nodes of the graph after processing them using a GCNN, while the second one uses a seq2seq architecture where the encoder uses the same GCNN as the first approach and the decoder is the one responsible of generating the layout sequentially taking into consideration the previously predicted objects and their bounding boxes to generate a new one.

4.1 SG2BB

The first system developed, called SG2BB is explained in two sections. First, an overall overview of the architecture is presented and finally, the matching system that matches the nodes of the graph with the objects (and their respective bounding boxes) of the ground truth (this step is only needed to train the model).

4.1.1 Architecture

The system developed follows the same architecture as SG2IM [Justin et al., 2018] but adding the following two key components:

- Graph generation from text (using AMR graphs).
- Matching algorithm of ground truth objects with nodes of the graph.

The main change compared to SG2BB, is that the new graphs include more relations between nodes that may add more meaningful information compared to only using geometric relationships. Moreover, the number of objects is increased as it includes objects that appear in the captions and not only the ones given by the dataset.

An overview of the architecture can be seen in Figure 4.1.

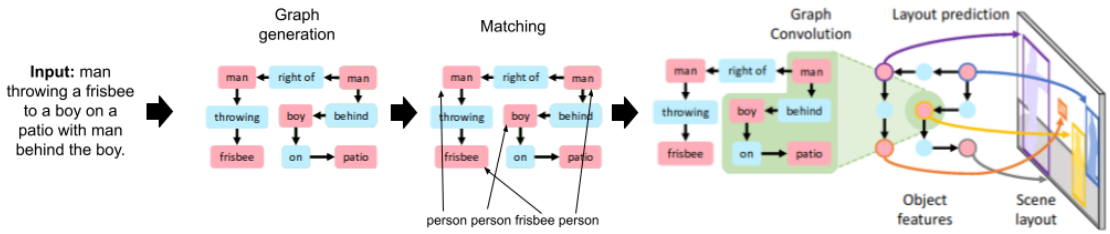


Figure 4.1: Overview of the SG2BB architecture.

First, a text is given as input to the network, which is transformed into a graph using AMR system. Once the graph is obtained, the matching algorithm, as explained in Section 4.1.2, is applied, which takes as input the nodes of the graph $O = \{o_1 ... o_n\}$ and the ground truth objects $GT = \{gt_1 ... gt_n\}$ and assigns the ground truth objects to the nodes. The matching algorithm is only needed during training time due to needing to specify explicitly which node of the graph belongs to each object (as a reference) because the nodes will be used to obtain the bounding boxes of the objects.

After defining the graphs, a graph convolutional neural network is used to process the graphs in an end-to-end manner. Given nodes and edges of size D_{in}, output vectors of size D_{out} are obtained as a function of a neighbourhood of their corresponding inputs so that each graph convolution layer propagates information along edges on the graph.

To calculate output vectors $v_i', v_r' \in \mathbb{R}^{D_{out}}$ the functions g_s, g_p, and g_o are used for every node and edge taking as input the triple of vectors (v_i, v_r, v_j) for an edge where $v_i, v_r \in \mathbb{R}^{D_{in}}$ for all objects $o_i \in O$ and edges $(o_i, r, o_j) \in E$. The resulting output creates new vectors for the subject o_i, relation r, and object o_j respectively.

Edges output vectors for v_r' are computed as $v_r' = g_p(v_i, v_r, v_j)$. Nevertheless, updating object vectors is more complicated, since an object o_i can participate in many relationships with other objects. To obtain v_i' for an object, o_i should depend on all vectors v_j for objects to which o_i is connected, as well as the relation vectors v_r that connects these. To this end, for each edge starting at o_i we use g_s to compute a candidate vector, collecting all such candidates in the set V_i^s; we similarly use g_o to compute a set of candidate vectors V_i^o for all edges terminating at o_i. Concretely,

$$V_i^s = \{g_s(v_i, v_r, v_j) : (o_i, r, o_j) \in E\} \tag{4.1}$$

$$V_i^o = \{g_o(v_i, v_r, v_j) : (o_j, r, o_i) \in E\} \tag{4.2}$$

The output vector for v_i' for object o_i is then computed as $v_i' = h(V_i^s \cup V_i^o)$ where h is a symmetric function which pools an input set of vectors to a single output vector. An example of the computational graph for a single graph convolution layer can be seen in Figure 4.2.

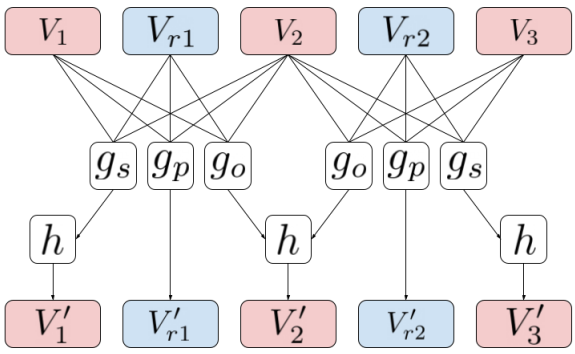


Figure 4.2: Computational graph illustrating a single graph convolution layer. The graph consists in two triples (o_1, r_1, o_2) and (o_3, r_2, o_2) with a total of three objects and two edges.

Finally, after processing the scene graphs a multilayer perceptron is used to generate the bounding boxes $b_i = (x_0, y_0, x_1, y_1)$ given the nodes embeddings, where (x_0, y_0) is the upper-left corner and (x_1, y_1) the lower-right corner.

In order to train the neural network Mean Squared Error loss (MSE) is used. This loss compares the ground truth bounding boxes (the ones assigned using the matching algorithm) with the ones predicted by the multilayer perceptron.

The results for this architecture can be found in Section 6.2.

4.1.2 Matching

The matching algorithm is one of the most important characteristics of this system. It works by taking into consideration the ground truth objects given by the dataset, which are matched with the nodes of the graph. Most of the times, the number of objects in the ground truth does not match the number of nodes in the graph, therefore, some objects or nodes need to be left unmatched. The matching process consists in calculating the cosine similarity between every object in the ground truth with every node in the graph and then maximize the similarity by matching each object with a node without repeating any of them. This problem is called the "linear sum assignment" also known as minimum weight matching in bipartite graphs.

Formally, let X be a boolean matrix where $X_{i,j} = 1$ if row i is assigned to column j. Then the optimal assignment has cost:

$$min \sum_{i}^{n} \sum_{j}^{m} C_{i,j} X_{i,j} \tag{4.3}$$

where each $C_{i,j}$ is the cost of matching vertex i with vertex j, that in our case is calculated by the cosine similarity. An example of the matrix $C_{i,j}$ can be seen in Table 4.1.

This is a minimization problem that is transformed into a maximization one in order to obtain the maximum similarity between objects and nodes.

Constraints of the problem:

1. The matrix only contains positive numbers.
2. If the matrix has more rows than columns then not every row needs to be assigned to a column. The same happens if there are more columns than rows.

GT / Nodes	Man	Kite	Frisbee	Plane
Airplane	0.1	0.2	0.4	0.8
Frisbee	0.3	0.4	1.0	0.7
Person	0.9	0.2	0.1	0.6

Table 4.1: Matching of ground truth objects with graph nodes. The cells in green are the matches. In this case, the node "kite" is left unmatched because there are not enough ground truth objects.

The steps followed to match objects and nodes are the following ones:

1. Each ground truth object and graph node is transformed to a word embedding, more specifically, GloVe embeddings [Pennington et al., 2014].

2. The cosine similarity between every object and node is calculated and the cost matrix C is generated.

3. The cosine similarity is in the range $[-1,1]$, therefore, each cell in matrix C is changed to range $[0,1]$ to satisfy the constraints. The formula used is: $new\_value_1 = old\_value \times 0.5 + 0.5$.

4. Invert all the values in matrix C to convert the problem into a maximization one. The formula used is: $new\_value_2 = 1 - new\_value_1$.

5. The problem is solved using the Hungarian algorithm [Kuhn., 1955].

To improve clarity, in Figure 4.3 we show how the system is able to correctly match MSCOCO objects [Lin et al., 2014] using a manually modified version of an AMR graph [Cai and Lam, 2020].

Caption: People are getting off a boat on a rocky island.
Graph triples: [[“people”, “getting off”, “boat”], [“people”, “on”, “island”]].
Nodes to match: people, boat, island.

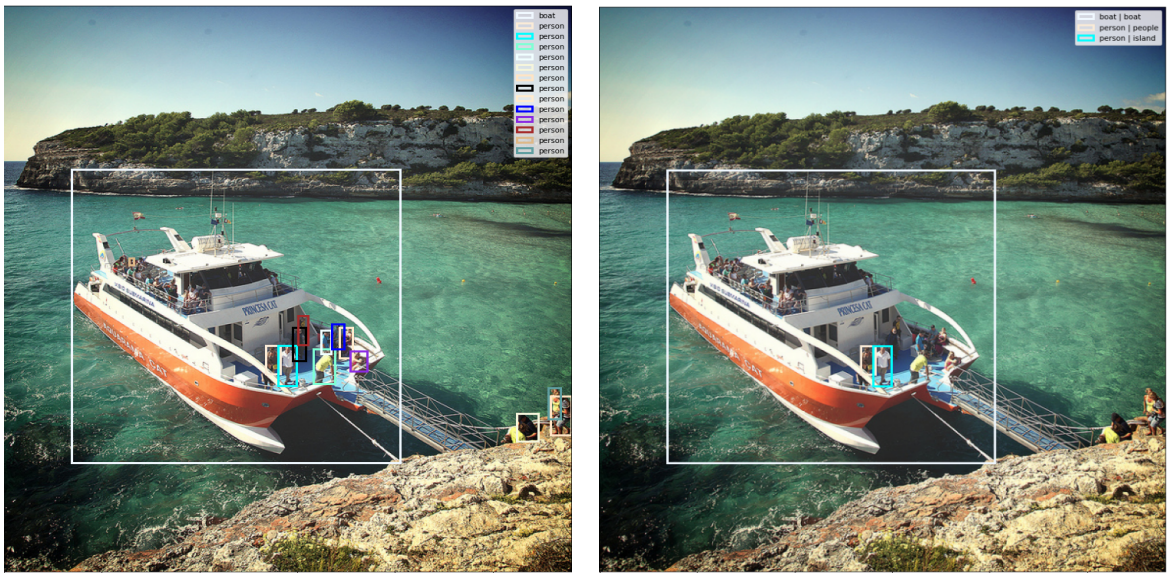


Figure 4.3: Comparison between ground truth objects (left) and the selected ones using the matching algorithm (right). The right image legend shows the matching between ground truth objects and graph nodes.

The biggest problem found using this algorithm is the loss of information as seen in Figure 4.3. The ground truth shows 13 persons but only one is matched correctly due to the graph having the node “People” to refer to all those persons. Moreover, the algorithm needs to match all the nodes, therefore, “island” is incorrectly matched with “person” due to being the only possibility left.

Using AMR graphs [Cai and Lam, 2020] has the problem that some nodes that do not have meaning like "play-01"may be matched. An example of an AMR graph, can be seen in Figure 3.4.

To understand how much information is lost when the matching algorithm is applied a new dataset has been developed using 100 different and randomly selected captions from the MSCOCO dataset. In this dataset, each ground truth object is matched with the most similar word that appears in the caption manually.

The matching algorithm obtains an accuracy of 33% for AMR graphs. This accuracy is lower than expected, which may happen due to the large vocabulary that the dataset has as seen in Table 4.2. The accuracy can be improved by adding a threshold of 0.7 that improves the accuracy up to 62% in AMR graphs. Nevertheless, adding a threshold to the algorithm incurs in having most of the times either one object or none for a given caption which makes the scene being simple.

	AMR	COCO
Number of objects:	18662	80
Number of relations:	85	None
Number of objects per MSCOCO objects:	233	1

Table 4.2: Number of objects in AMR graphs compared to number of objects in MSCOCO.

The number of object categories that MSCOCO dataset has is not enough, as each object needs to match on average 233 different objects from AMR objects, which is almost impossible.

4.2 GCN2LY

The second system developed is based on Obj-GAN [Li et al., 2019b] and we call it GCN2LY. We made some modifications over the original architecture. On one hand, the encoder is changed from an RNN to a GCNN to explore if a structured version of the text can contribute to better represent the text. On the other hand, the decoder is changed from using a gaussian mixture to a grid-based system, which is simpler to implement and more flexible to adapt to the requirements of the dataset. An overview of the system can be seen in Figure 4.4.

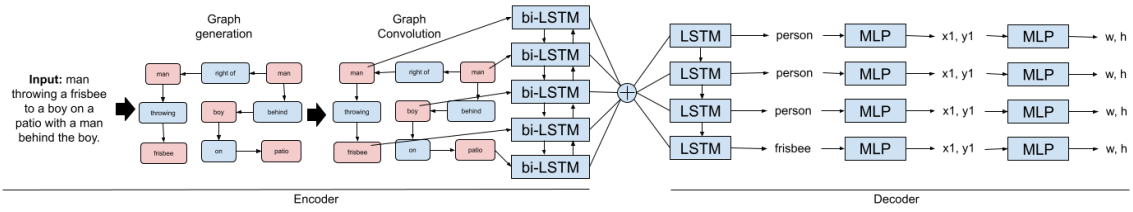


Figure 4.4: Overview of the GCN2LY architecture.

The system uses a seq2seq architecture that includes an encoder that transforms the graph into a hidden representation and a decoder that using the hidden representation sequentially generates the objects and their respective bounding boxes.

4.2.1 Encoder

First, a text is given as input to the network which is transformed into a graph using the AMR [Cai and Lam, 2020] system. After defining the graphs, a graph convolutional neural network is used to process the graphs in an end-to-end manner where the output vectors are a function of a neighbourhood of their corresponding inputs so that each graph convolution layer propagates information along edges on the graph. Once the graph is processed the node embeddings are again processed using a bidirectional LSTM (bi-LSTM) [Hochreiter and Schmidhuber, 1997], to compress all these embeddings into a hidden representation vector. For better explanations about how the GCNN works the reader is referred to Section 4.1.1.

4.2.2 Decoder

The decoder uses an LSTM to sequentially generate at every step t a hidden-state and an output-state vector which are going to be used to generate the next object and bounding box.

Let the output of the decoder at each time-step be $O_t = (b_t, l_t)$, where $b_t = (x_1, y_1, w, h)$ is a bounding box with four parameters: (x_1, y_1) refer to the center of the bounding box and (w, h) to the width and height of the bounding box respectively. l_t is the object category to be drawn given a set of objects C.

At every time-step t, O_t is generated in the following way:

- First, using the bounding box and object category obtained in the previous step O_{t-1}, a forward pass is done through the LSTM obtaining an output-state and a hidden-state vector. The exception is $t = 1$, where $l_0 = 1$ and $b_0 = (0, 0, 0, 0)$ referring to the " start of sequence <sos> " token.

- Then, l_t is obtained using a probability distribution with the probability for each object to be drawn. This is done by introducing the output-state of the LSTM in a dense layer. The probability distribution to obtain l_t is a vector of size C where C is the size of the set of given object categories and where each position in the vector is a real number specifying the probability for an object to be drawn. The object to draw l_t is the index with the maximum value.

- After obtaining l_t, (x_1, y_1) is obtained using a probability distribution with the probability for an object to be drawn in a specific position. This is done by introducing the hidden-state of the LSTM and the one-hot encoding vector of the object to be drawn l_t to an MLP. The probability distribution to obtain (x_1, y_1) is a vector of size N^2, representing the size of the grid. Each position in the vector is the probability for an object to be drawn in the upper-left corner that is referring. The position (x_1, y_1) is obtained by sampling it from a multinomial probability distribution. For better explanations about how the grid system works the reader is referred to Section 4.2.2.

- Finally, to obtain the width w and height h an MLP is used using the previous sampled position (x_1, y_1), the one-hot encoding vector of the object to be drawn l_t, and the hidden-state of the LSTM.
- This process is applied again until a maximum number of steps is achieved or when "end of sequence <eos>" or "padding <pad>" token is obtained. On validation time, and inspired on Obj-GAN, we filter redundant labels. This is done by sampling an upper-bound threshold from a normal distribution using each object category's mean and standard deviation. The sampled value is used to select the objects that we are going to keep.

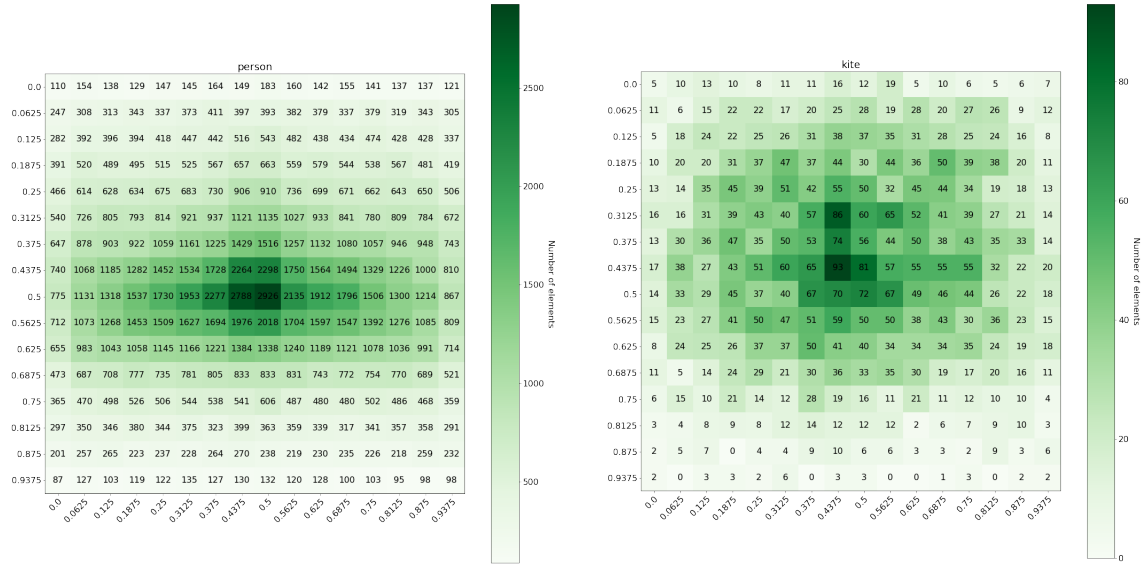
To train the neural network three losses are used: categorical cross-entropy for the object category, categorical cross-entropy for (x_1, y_1) position, and Mean Squared Error (MSE) for width w and height h. Moreover, when generating the next object, attention and teacher forcing are applied when training the model, therefore, at time-step t the real values of time-step $t-1, GT_{t-1} = (b_{t-1}, l_{t-1})$ and the global sentence information are used. The attention mechanism [Bahdanau et al., 2015] takes the previous hidden-state $h_{(t-1)}$ of the decoder, and all of the stacked forward and backward hidden-states from the encoder. The layer will output an attention vector, a_t, that is the length of the encoder output, where each element is between 0 and 1, and the entire vector sums 1. Intuitively, this layer pays attention to the most relevant nodes of the input in order to correctly predict the next object.

The results obtained using this architecture can be found in Section 6.2.

Decoder grid system

As shown in Section 6.4, SG2BB system draws all the objects in the center of the picture with a bounding box that resembles a square. The same system was tried for GCN2LY using the MLP after the LSTM in the decoder for each time-step, which gave similar results, therefore, there was a need to develop a new system that draws the objects in different locations with different shapes and sizes.

A careful review of the dataset shows that most of the objects bounding box centers, as shown in Figure 4.5a and Figure 4.5b, are located in the center of the image and we think that it may lead the neural network to guess that the optimal position in respect to the MSE is always in the center of the picture.



(a) Location of the centers of all person bounding boxes.

(b) Location of the centers of all kite bounding boxes.

Figure 4.5: All person and kite bounding boxes centers are located in the center of the picture. Grid of size 16x16.

The neural network guesses that the optimal position for placing all the bounding boxes is in the middle due to most of the centers being located there, which leads to a layout that is not coherent with the text that describes it. To solve this problem, a probabilistic component inspired by Obj-GAN [Li et al., 2019b] was developed.

Obj-GAN uses a gaussian mixture model in order to sample the bounding boxes. In our case, we propose an original approach that consists in the creation of a grid matrix M of size (N,N) as seen in Figure 4.6.

The grid can divide an image into sections of the same size, for example, a 4x4 grid contains 16 sections with a width and height of 0.25, considering that the image size is in the range $[0,1]$. Each position has a representative upper-left corner, for example, section 1 upper-left corner has position $X = 0, Y = 0$ and section 10 has position $X = 0.50, Y = 0.25$.

Using the intuition mentioned above, the grid can be considered as a discrete probability distribution where each position is the probability to draw an object there. This probability distribution can be learnt using a neural network and then, a position can be sampled using a multinomial probability distribution. The upper-left corner of the sampled position can be used as (x_1, y_1) (center) of the bounding box for a given object.

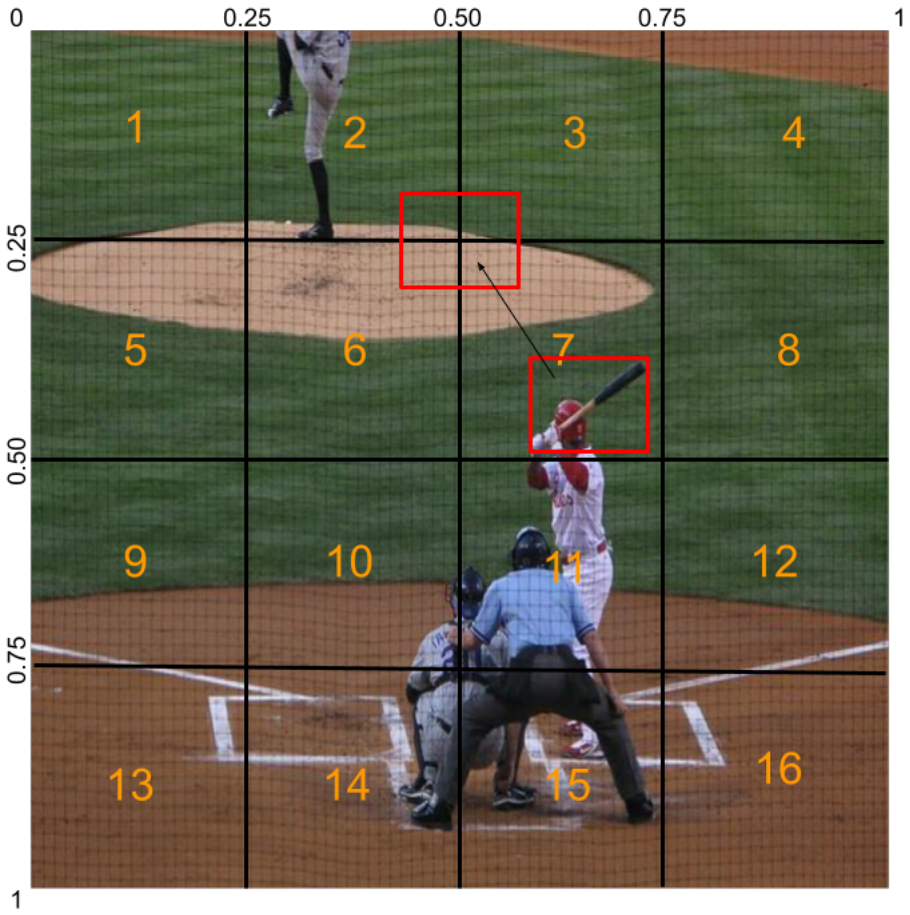


Figure 4.6: Grid system of size 4x4. The ground truth objects are moved from their original positions to the closest upper-left corner.

Using this approach, all ground truth objects need to be moved to the closest upper-left corner which causes an error in their position. Nevertheless, this error can be minimized using a bigger grid, for example, a 4x4 matrix has an error of 0.25 that in pixels is 64 pixels for a 256×256 pixels image, therefore, each bounding box is moved 64 pixels to the left and the top in the worst-case scenario. If a 32x32 matrix is used the error is reduced to 0.03125 that in pixels is 8 pixels for 256×256 pixels image.

4.3 RNN2LY

An additional third system based on GCN2LY has been developed where the encoder is changed from a GCNN to an RNN and the same decoder is maintained. This system has been developed in order to see and compare if a structured version of the text contributes to improve the quality of the generated layout. An overview of the system can be found in Figure 4.7. The RNN used is the one that Obj-GAN [Li et al., 2019b] uses.

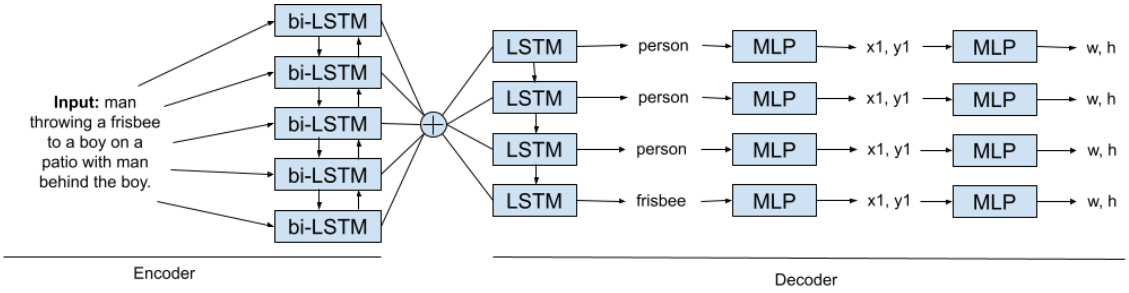


Figure 4.7: Overview of the RNN2LY architecture.

CHAPTER 5

Metrics to measure the quality of layouts

In this chapter, we will discuss about the need of developing new metrics due to the ones currently in use do not measure correctly the quality of the generated layouts. In Section 5.2 new metrics are proposed that better measure the spatial composition of scenes.

5.1 The necessity of developing new metrics

Nowadays, in order to measure the quality of bounding boxes, the Intersection over Union (IoU) metric [Everingham et al., 2014] is provided. This metric, allows to measure the overlap between the predicted and the ground truth bounding boxes and for object detection task, the precision and recall metrics are calculated using the IoU value for a given IoU threshold, these are called precision@IoU and recall@IoU respectively. Research papers tend to give the results for the MSCOCO dataset [Lin et al., 2014] with a threshold that goes from 0.5 to 0.95 with a step size of 0.05 and then calculating the average between all the thresholds, also called mean average precision (mAP) and mean average recall (mAR). These metrics, however, may not be suitable for the task of text-to-layout, for that reason, there is a need for developing new metrics that are suitable and better measure the spatial composition of scenes. In Figure 5.1 an example of why the IoU fails to measure the quality of the layout is shown.

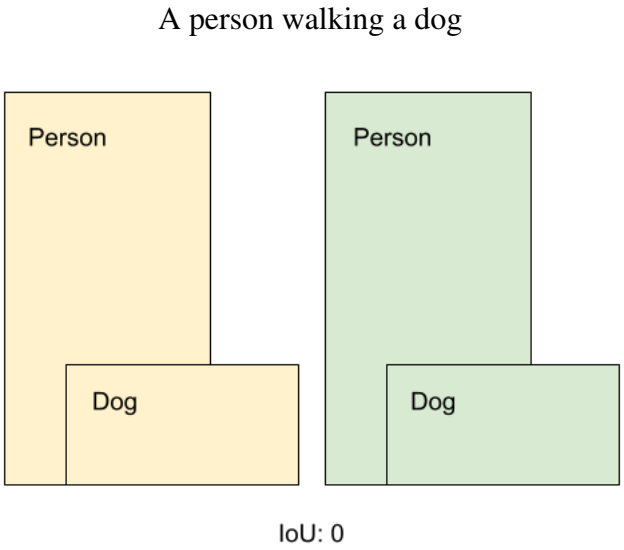


Figure 5.1: The ground truth bounding boxes (yellow) and the predicted bounding boxes (green) in the same picture. The predicted layout is the same as the ground truth layout but displaced to the right. Nevertheless, even if the text description matches the predicted layout (and the ground truth layout in sizes), the IoU is 0 because there is no overlap with the ground truth layout and therefore, the predicted layout is considered incorrect.

5.2 Proposed metrics

The following metrics proposed have been developed in order to better measure the spatial composition of scenes and are inspired on [Tan et al., 2019] and [Tripathi et al., 2019] papers.

5.2.1 Relative spatial categorical position

This metric allows to identify whether an object is located to the *left, right, above,* or *below* with respect to another object. To calculate this metric every pair of objects is taken into consideration and two steps are performed. First, their respective bounding boxes centers are calculated and then four imaginary lines are thrown from the center of one of the two objects with the angles of $\frac{\Pi}{4}$, $\frac{3\Pi}{4}$, $\frac{5\Pi}{4}$, and $\frac{7\Pi}{4}$ as shown in Figure 5.2. Depending on where the second object center lies the spatial categorical position is decided as follows:

- Between $\frac{\Pi}{4}$ and $\frac{-\Pi}{4}$ the second object is to the right of the first one.
- Between $\frac{\Pi}{4}$ and $\frac{3\Pi}{4}$ the second object is above the first one.
- Between $\frac{3\Pi}{4}$ and $\frac{5\Pi}{4}$ the second object is to the left of the first one.
- Between $\frac{5\Pi}{4}$ and $\frac{7\Pi}{4}$ the second object is below the first one.

This metric is calculated for every pair of ground truth and predicted bounding boxes considering the right-left symmetry (left and right are considered the same position), then if both pairs (ground truth and the prediction) match the same spatial categorical position the prediction is considered correct, otherwise incorrect. This metric lies in between $[0, 1]$ taking the mean into account, therefore, for a given layout, a 1 means that all predicted bounding boxes are well located with respect to the ground truth bounding boxes.

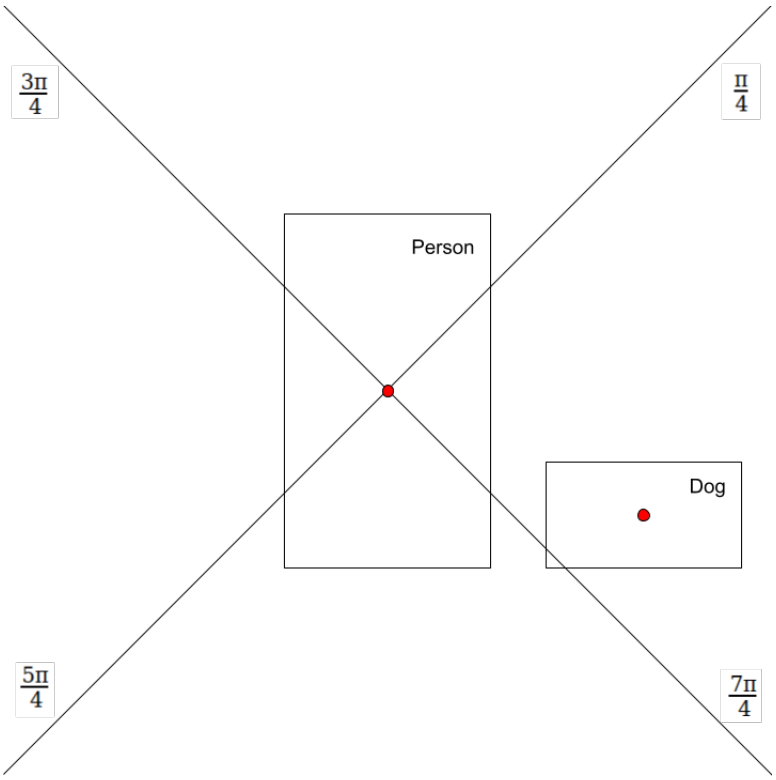


Figure 5.2: Visual representation of how the relative spatial categorical position between person and dog is obtained. Red points are the center of the bounding boxes for each object. The dog is located on the right side of the person as its center lies between $\frac{\Pi}{4}$ and $\frac{-\Pi}{4}$.

5.2.2 Aspect ratio

This metric allows to identify whether the predicted bounding boxes maintain the same aspect ratio (width and height ratio) with respect to the ground truth bounding boxes. To calculate this metric, for each predicted bounding box BB_p, the ground truth bounding box BB_{gt} is retrieved, then the internal angle of both bounding boxes are calculated α_p and α_{gt} in the range $[0, \frac{\Pi}{2}]$ using the coordinates of both corners (left-bottom and top-right). Finally the aspect ratio is given by $|\alpha_p - \alpha_{gt}|$, also in the range $[0, \frac{\Pi}{2}]$. An example of how the metric is calculated can be seen in Figure 5.3.

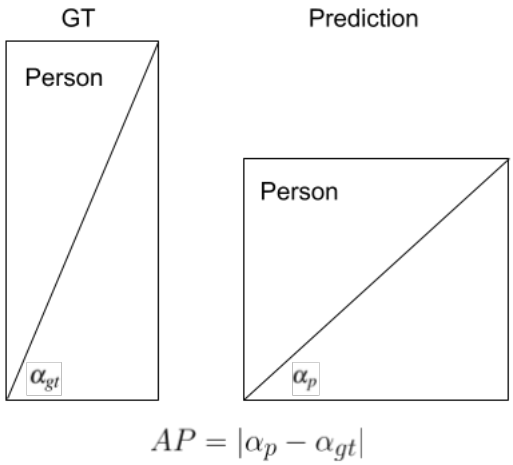


Figure 5.3: Aspect ratio between ground truth and predicted object.

Depending on the height and width of the bounding boxes the internal angle can have the following different values:

- If the bounding box is taller than wider, its internal angle is bigger than $\frac{\Pi}{4}$.
- If the bounding box is as tall as wide, its internal angle is $\frac{\Pi}{4}$.
- If the bounding box is wider than taller, its internal angle is smaller than $\frac{\Pi}{4}$.

This metric is calculated for each ground truth and predicted bounding boxes which lies in between $[0, \frac{\Pi}{2}]$ taking the mean into account, therefore, if the obtained score is 0, the prediction perfectly match the aspect ratios of the ground truth bounding boxes and the closer the score gets to $\frac{\Pi}{2}$ the worse the bounding boxes are predicted.

5.2.3 Class matching

This metric allows to identify whether the predicted objects are the same to those that appear in the ground truth. For that purpose, the precision, recall, and F1-score [Powers, 2019] are calculated.

Considering the following terminology:

- **True Positive (TP)**: The ground truth class and the predicted class are the same.
- **False positive (FP)**: The predicted class does not appear in the ground truth.
- **False negative (FN)**: The ground truth class does not appear in the prediction.

The precision, recall, and F1-score meaning and equations are the following:

- **Precision (P):** The precision is the amount of correctly predicted classes that appear in the ground truth and is calculated as:

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

- **Recall (R):** The recall is the amount of correctly predicted classes compared to all the ground truth classes and is calculated as:

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

- **F1-Score (F1):** The F1-score is a weighted average of precision and recall and is calculated as:

$$F1\text{-}score = \frac{2 * recall * precision}{recall + precision} \tag{5.3}$$

5.2.4 Relative scale difference

This metric allows to identify whether the relative sizes of predicted and ground truth bounding boxes are similar. For example, given two bounding boxes, person and phone, no matter how big the person is, a phone will be smaller. To calculate this metric, first, for every pair of ground truth bounding boxes, their respective pair of predicted bounding boxes are retrieved. For each bounding box, the areas A_{p1}, A_{p2}, A_{gt1}, and A_{gt2} are calculated, then the ratio of predicted areas and ground truth areas are calculated as $R_p = \frac{A_{p1}}{A_{p2}}$ and $R_{gt} = \frac{A_{gt1}}{A_{gt2}}$. The absolute value between $|R_p - R_{gt}|$ is the error for a pair.

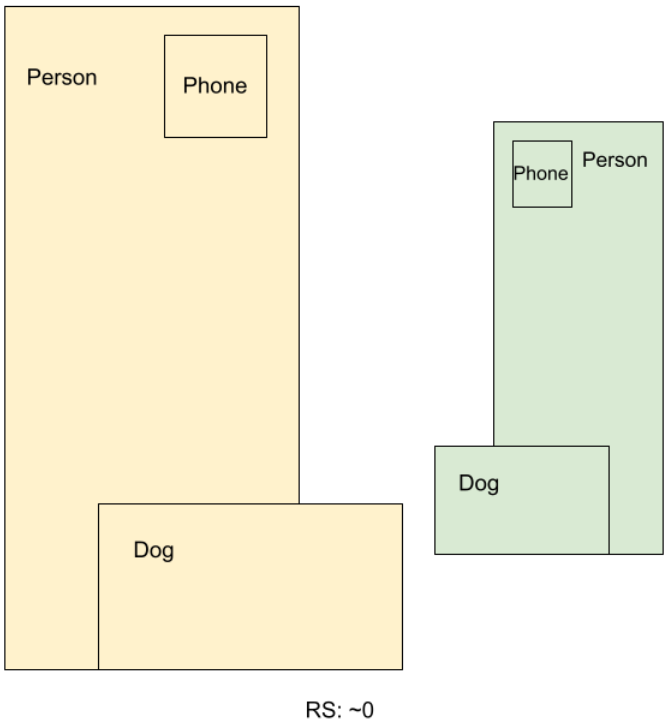


Figure 5.4: Relative scale difference between ground truth objects (green) and predicted objects (yellow). In this example, the predicted objects are scaled-down and moved compared to the ground truth objects. The relative scale is ≈ 0 because the relative areas between the objects are the same.

This metric is calculated for every pair of ground truth and predicted bounding boxes. Nevertheless, it does not have a defined maximum value, therefore, if the obtained score is 0, the predicted bounding boxes perfectly match the relative scale difference of the ground truth bounding boxes, and the bigger the value the worse are predicted.

CHAPTER 6

Experiments and results

In this chapter, the experiments and results of the architectures developed in this project are covered. Before providing the final results the experimental setup is described, which includes the partitioning of the dataset and the model selection criterion. Afterwards, a deep analysis of the results is provided.

6.1 Experimental setup

6.1.1 Dataset partitions

The dataset selected for training and evaluating all the models is the MSCOCO 2014 dataset [Lin et al., 2014] which is divided into three partitions: training, development, and testing. As the test set annotations are not publicly available, we use the development set as our test set. This decision forces us to create a new development set. For that purpose, the training partition has been further divided into two sets: the 90% of the contents of the initial training partition (74504 pictures) form the new train set, while the remaining 10% of the contents (8279 pictures) form the new development set. The testing partition (i.e. the original development set) is maintained without any modifications (40504 pictures).

6.1.2 Model selection

Model selection has been done using the new metrics proposed due to the losses not providing enough information to measure the quality of the model as seen in Section 6.2. The metrics used for model selection are the following: relative spatial categorical position (RSCP), aspect ratio (AR), relative scale (RS), precision (P), F1-score (F1), and recall (R). These metrics have been all aggregated according to Equation 6.1.

$$score = RSCP + (1 - AR) + (1 - softmax(RS)) + P + F1 + R \tag{6.1}$$

As AR and RS assign a lower value to better performance, we invert them for the aggregation. Moreover, RS is the only metric with an undefined maximum, therefore, the softmax function is applied in order to normalize it in the range $[0, 1]$ using the RS of all the epochs.

The epoch with the highest score in the development dataset is the selected model to be evaluated in the testing partition.

6.2 Results

In the following subsections, specific details of how each system is trained are provided. The graphs used as input of the system are generated using the AMR [Cai and Lam, 2020] system.

6.2.1 SG2BB

SG2BB, explained in Section 4.1, has been trained over 10 epochs using Adam optimizer with a learning rate of $1e^{-4}$ and using a hidden size of 128 for the GCNN. In Figure 6.1 the evolution of the loss can be found.

Figure 6.1 shows that the system stalls fast after the first 2 epochs without any more improvement and starts overfitting after the fourth epoch. Nevertheless, the metrics proposed show a slight improvement in every epoch as seen in Figure 6.2.

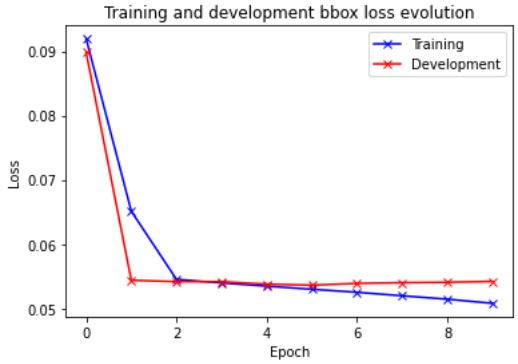


Figure 6.1: SG2BB system MSE loss of bounding boxes.

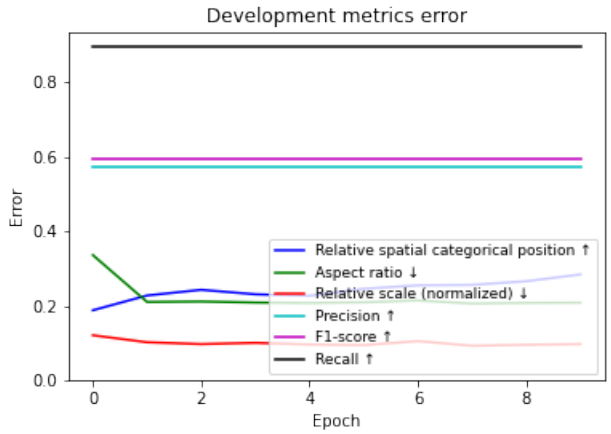
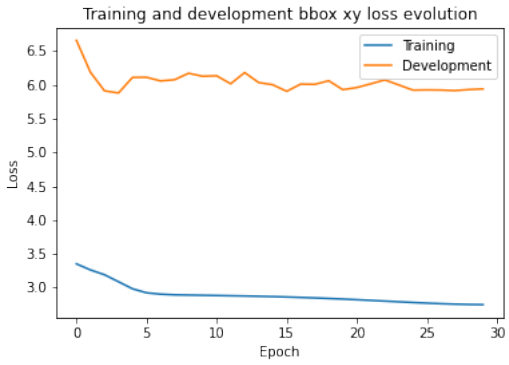


Figure 6.2: SG2BB metrics evolution (development partition).

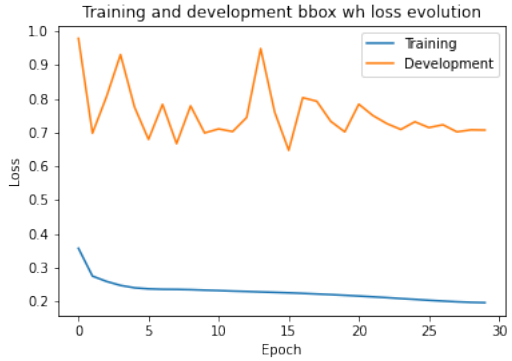
6.2.2 GCN2LY

GCN2LY, explained in Section 4.2, has been trained over 30 epochs using Adam optimizer, one cycle LR scheduler with a learning rate of $1e^{-3}$, weight decay of $1e^{-4}$, and a hidden size of 256 for the encoder/decoder. This system uses the 10 biggest ground truth objects by area and a grid system of 32×32. In Figure 6.3 the evolution of the losses used can be found.

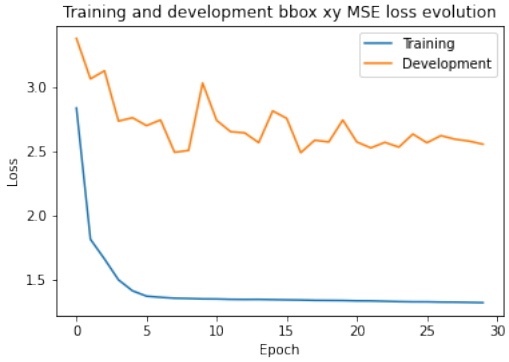
(a) Categorical cross-entropy loss for object category.



(b) Categorical cross-entropy loss for x and y (center of the bounding box).



(c) MSE loss for width and height of the bounding box.



(d) MSE loss for x and y (center of the bounding box).

Figure 6.3: Figures showing the training and development losses for GCN2LY system.

Figure 6.3 shows that there is a high difference between training and development sets values with the development one not improving too much through time. Nevertheless, the metrics used to evaluate the system show that there is an improvement in every epoch as seen in Figure 6.4, therefore, the losses do not give insightful information about the real performance of the model and the metrics are needed to measure the quality of it.

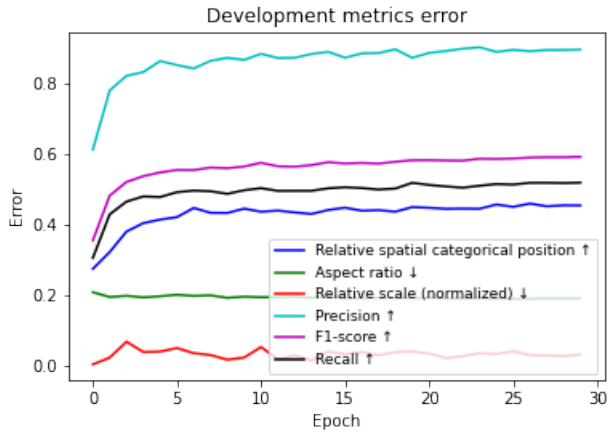
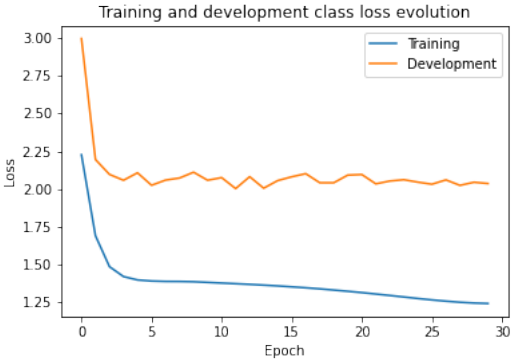


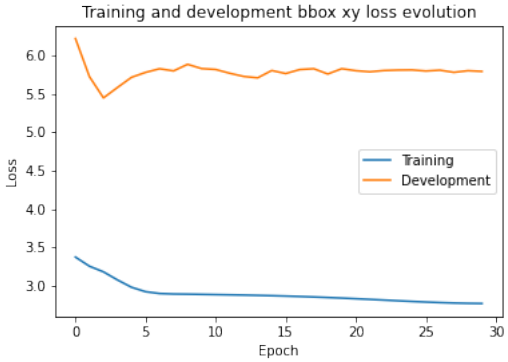
Figure 6.4: GCN2LY metrics evolution (development partition).

6.2.3 RNN2LY

RNN2LY, explained in Section 4.3, has been trained over 30 epochs using Adam optimizer, one cycle LR scheduler with a learning rate of $1e^{-3}$, weight decay of $1e^{-4}$, and a hidden size of 256 for the encoder/decoder. This system uses the 10 biggest ground truth objects by area and a grid system of 32×32. Moreover, three versions of the encoder have been trained: training the RNN with a random initialization of its weights, using a pretrained RNN (the same one that Obj-GAN [Li et al., 2019b] uses) and keeping it frozen during the training of the decoder, and fine-tuning the pretrained RNN in the target task. The evolution of the losses for each system can be found in Figures 6.5, 6.6, and 6.7 respectively.

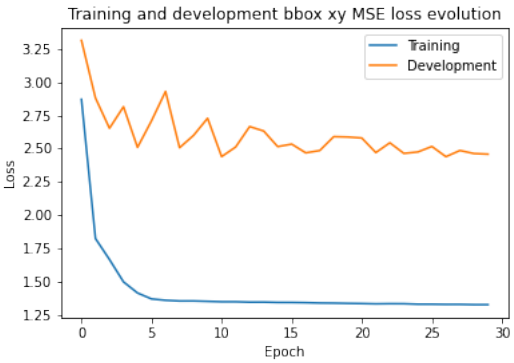


(a) Categorical cross-entropy loss for object category.



(b) Categorical cross-entropy loss for x and y (center of the bounding box).

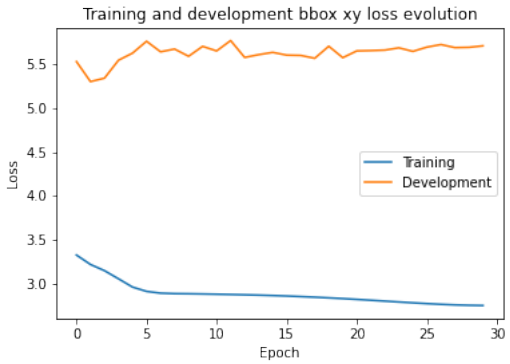
(c) MSE loss for width and height of the bounding box.



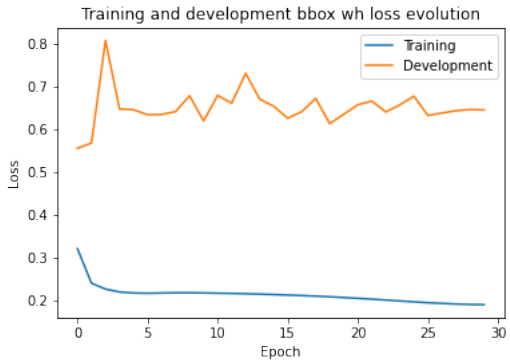
(d) MSE loss for x and y (center of the bounding box).

Figure 6.5: Figures showing the training and development losses for RNN2LY system with the randomly initialized encoder.

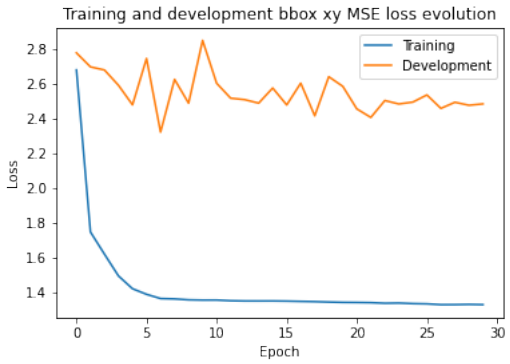
(a) Categorical cross-entropy loss for object category.



(b) Categorical cross-entropy loss for x and y (center of the bounding box).



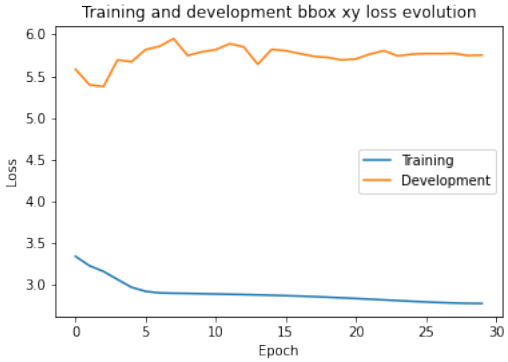
(c) MSE loss for width and height of the bounding box.



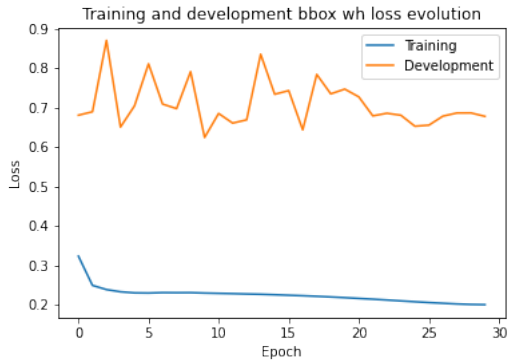
(d) MSE loss for x and y (center of the bounding box).

Figure 6.6: Figures showing the training and development losses for RNN2LY system with the pretrained and frozen RNN encoder.

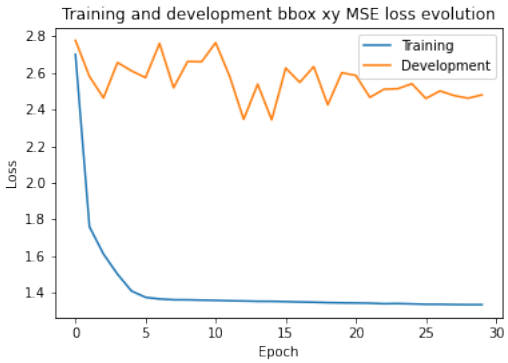
(a) Categorical cross-entropy loss for object category.



(b) Categorical cross-entropy loss for x and y (center of the bounding box).



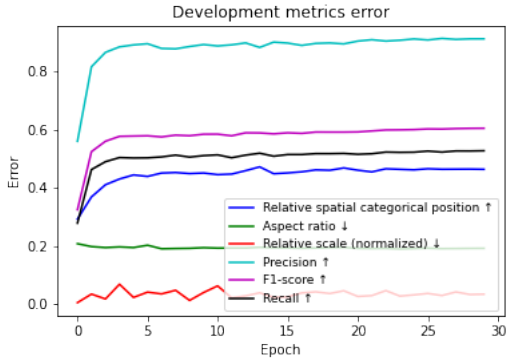
(c) MSE loss for width and height of the bounding box.



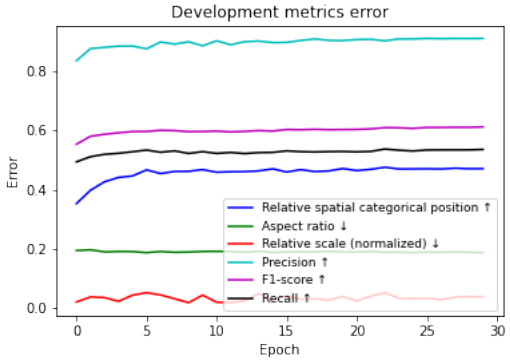
(d) MSE loss for x and y (center of the bounding box).

Figure 6.7: Figures showing the training and development losses for RNN2LY system with the pretrained but not frozen RNN encoder.

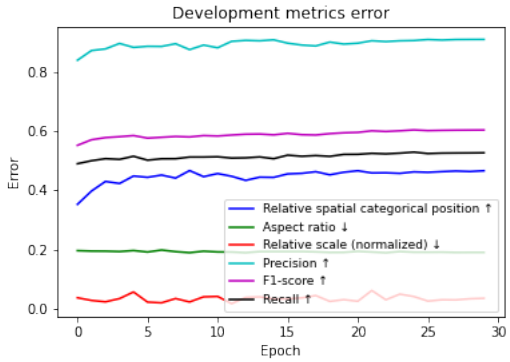
Figures 6.5, 6.6, and 6.7 show that there is a high difference between training and development sets values with the development one not improving too much through time. Nevertheless, similar to what happens with GCN2LY system, the metrics used to evaluate the systems show that there is an improvement in every epoch as seen in Figure 6.8, therefore, the losses do not give insightful information about the real performance of the model and the metrics are needed to measure the quality of it.



(a) RNN2LY metrics with randomly initialized weights.



(b) RNN2LY metrics with a pretrained and frozen RNN encoder.



(c) RNN2LY with a pretrained but not frozen RNN encoder.

Figure 6.8: RNN2LY metrics evolution (development partition).

6.3 Comparison of the developed systems

In Table 6.1, the comparison between the developed systems is shown. Additionally, we also add the results of a state-of-the-art system as a strong baseline: Obj-GAN[1]. Table 6.1 depicts the following metrics for each system obtained in our test partition: relative spatial categorical position (RSCP), aspect ratio (AR), relative scale (RS), precision (P), F1-score (F1), recall (R), precision with an IoU threshold of 0.3 and 0.5, and recall with an IoU threshold of 0.3 and 0.5.

System	RSCP ↑	AR ↓	RS ↓	P ↑	F1 ↑	R ↑	P@0.3 ↑	P@0.5 ↑	R@0.3 ↑	R@0.5 ↑
Obj-GAN	0.348	0.246	2216.491	0.866	0.566	0.499	0.257	0.094	0.227	0.073
SG2BB	0.279	0.208	14.750	0.891†	0.594†	0.578†	0.190	0.073	0.080	0.032
GCN2LY	0.449	0.191	7.540	0.893	0.594	0.520	0.362	0.172	0.286	0.159
RNN2LY	0.459	0.191	7.503	0.910	0.606	0.529	0.360	0.171	0.295	0.165
RNN2LY$_{PT}$	**0.464**	**0.188**	7.587	0.907	**0.609**	**0.534**	0.358	0.171	0.291	0.164
RNN2LY$_{FT}$	0.459	0.190	**7.478**	**0.911**	0.606	0.529	**0.364**	**0.173**	**0.296**	**0.167**

Table 6.1: Results obtained with several systems on our test partition for the text-to-layout task. † indicates that those results are distorted by the heuristic matching algorithm and thus not very significant. PT indicates that the model uses a pretrained encoder. FT indicates that the model uses a pretrained and fine-tuned encoder.

Table 6.1 shows that GCN2LY and RNN2LY improve all the metrics significantly compared to the baseline system Obj-GAN [Li et al., 2019b].

Unfortunately, the use of a structured version of the text does not incur in any improvement of the generated layout compared to using plain text as seen with RNN2LY system due to providing similar metrics and visual layouts as seen in Section 6.4. Nevertheless, we want to remark that when converting text to graphs there is some information loss, therefore, GCN2LY system could be better if this noise is reduced or removed.

We want to note that the relative scale metric for Obj-GAN system is not stable due to generating too many small objects. Moreover, SG2BB precision, F1-score, and recall metrics may be misleading because we use the matching algorithm itself to calculate the values. In that sense, the provided results are an upper-bound and thus not very significant.

[1]The best execution among 10 has been taken for comparison due to high variations in the output.

For GCN2LY and RNN2LY systems, in order to calculate the metrics, we match the predicted objects with the ground truth objects by maximizing the IoU. The matching is required as we do not have any reference object for each predicted one, and this can be done using the matching algorithm proposed for SG2BB system. An example of this matching can be seen in Figure 6.9.

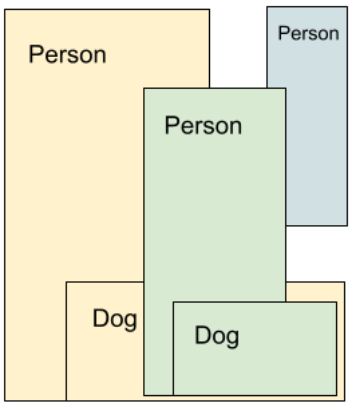


Figure 6.9: The ground truth objects (yellow), the predicted and matched objects (green), and the predicted but not matched objects (blue). Each predicted object is matched with the ground truth object that overlaps the most taking into account the categories. When there are more predicted or ground truth objects, these are left unmatched.

6.4 Qualitative analysis of the results

In this section, a qualitative comparison of the obtained layouts using the previous systems is shown.

6.4.1 SG2BB

(a) A dog looking up in at a frisbee. COCO image ID: 79407.

(b) A tennis player contorts his body to make contact with the ball. COCO image ID: 520478.

Figure 6.10: Figures showing the layout that SG2BB systems produces.

SG2BB system is not able to generate complex layouts as it places all the objects in the center of the image, therefore, the relations specified in the captions are not maintained. For example, in Figure 6.10a, the relation "looking up" appears in the caption, but SG2BB places the "frisbee" in the middle of the picture. Moreover, it produces objects that do not make sense, as seen in Figure 6.10b, where the system extracts the object "body", but this one is already placed as "person", therefore, the object is repeated twice. The relative sizes are not learned also. For example, in Figure 6.10a, the "frisbee" is too big compared to the "dog".

Taking into consideration all the previous points and the results in Table 6.1, we think that SG2BB is not a suitable system for the text-to-layout task.

6.4.2 Obj-GAN vs. GCN2LY and RNN2LY

In this section, the comparison between the ground truth, Obj-GAN, GCN2LY, and RNN2LY is done. Given that the different variants of RNN2LY are very similar, we only show the layouts obtained with the randomly initialized encoder.

Caption: A dirt covered floor in a home kitchen.

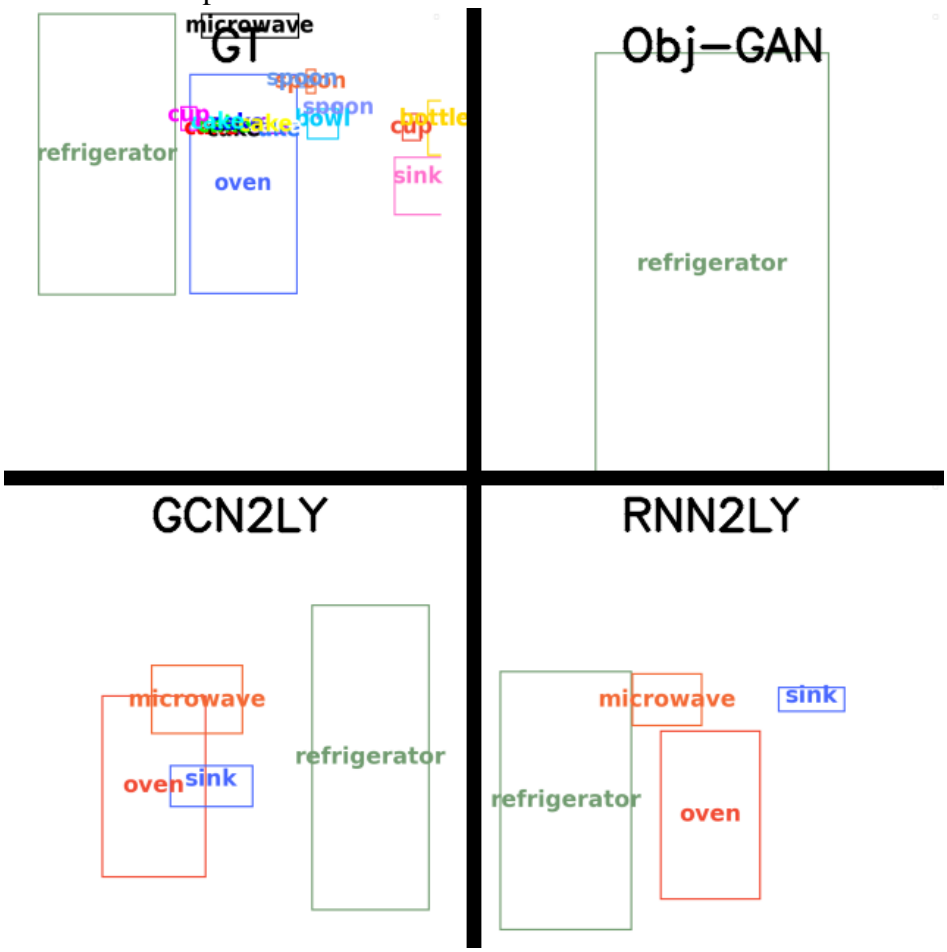


Figure 6.11: A dirt covered floor in a home kitchen. MSCOCO image ID: 65358.

Figure 6.11, shows how the ground truth layout has many different objects, small and big, related with "kitchen". The three systems, Obj-GAN, GCN2LY, and RNN2LY are able to extract objects that do not appear in the caption, but that are related with the concept of "kitchen". Nevertheless, Obj-GAN fails to produce a complex layout due to obtaining only a "refrigerator" object. In the case of GCN2LY and RNN2LY, both of them are able to build complex layouts using the most important objects, with RNN2LY being better than GCN2LY, due to correctly placing the "sink" separated from the "oven".

Caption: A woman sitting with a boy cutting a cake.

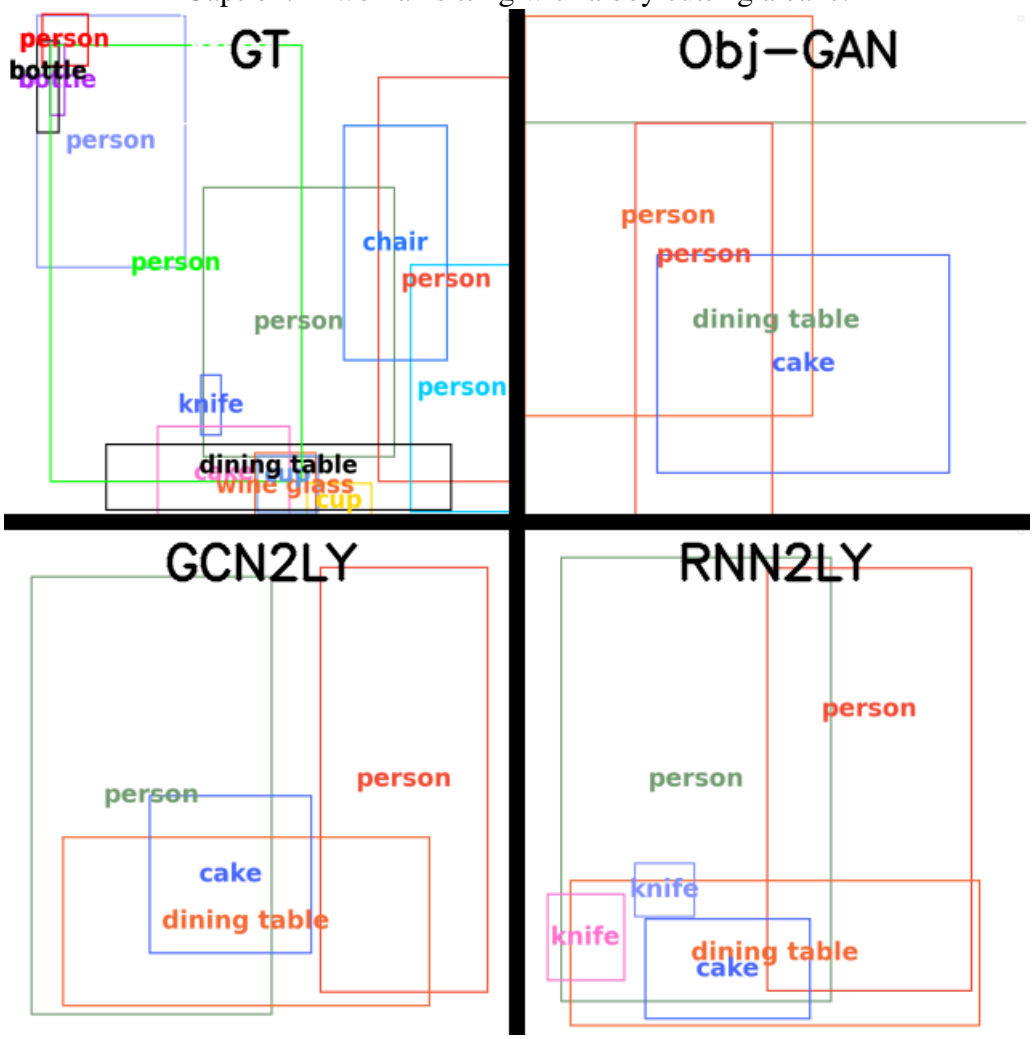


Figure 6.12: A woman sitting with a boy cutting a cake. MSCOCO image ID: 194097.

Figure 6.12, shows how the ground truth layout has many different objects, mainly big ones. The three systems, Obj-GAN, GCN2LY, and RNN2LY, are able to extract objects that appear in the caption and related ones, such as dining table. Moreover, we can see how GCN2LY and RNN2LY produce similar layouts that are more visually attractive compared to Obj-GAN.

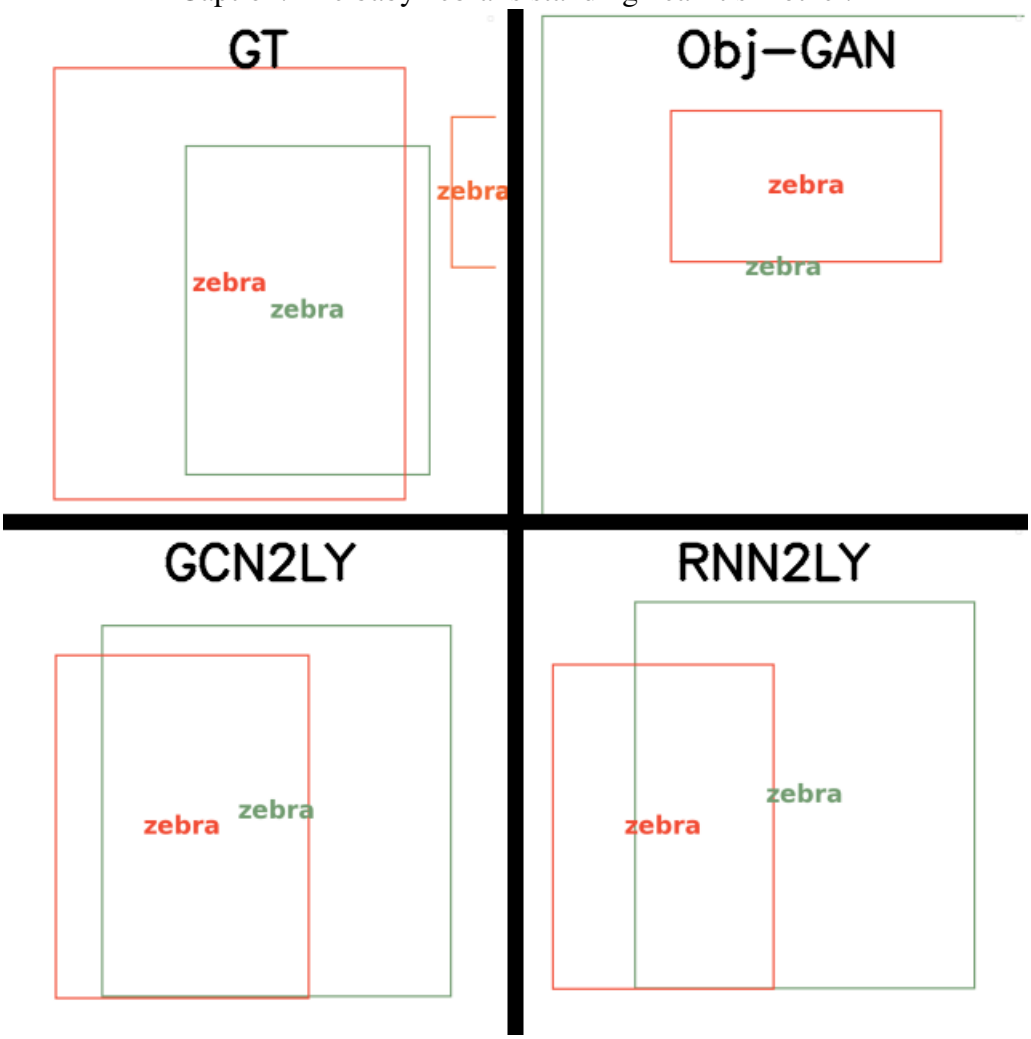


Figure 6.13: The baby zebra is standing near it's mother. MSCOCO image ID: 23411.

Figure 6.13, shows how the three systems, Obj-GAN, GCN2LY, and RNN2LY, are able to extract two zebras as specified in the caption. Nevertheless, Obj-GAN is unable to maintain the relative sizes between the objects as one is surrounding the other. For GCN2LY and RNN2LY, the relation "standing near" is achieved, as the zebras are placed close from each other. Moreover, one zebra is smaller than the other, making the layout correct due to one of them being a "baby zebra".

Caption: A man sliding into a base next to another baseball player.

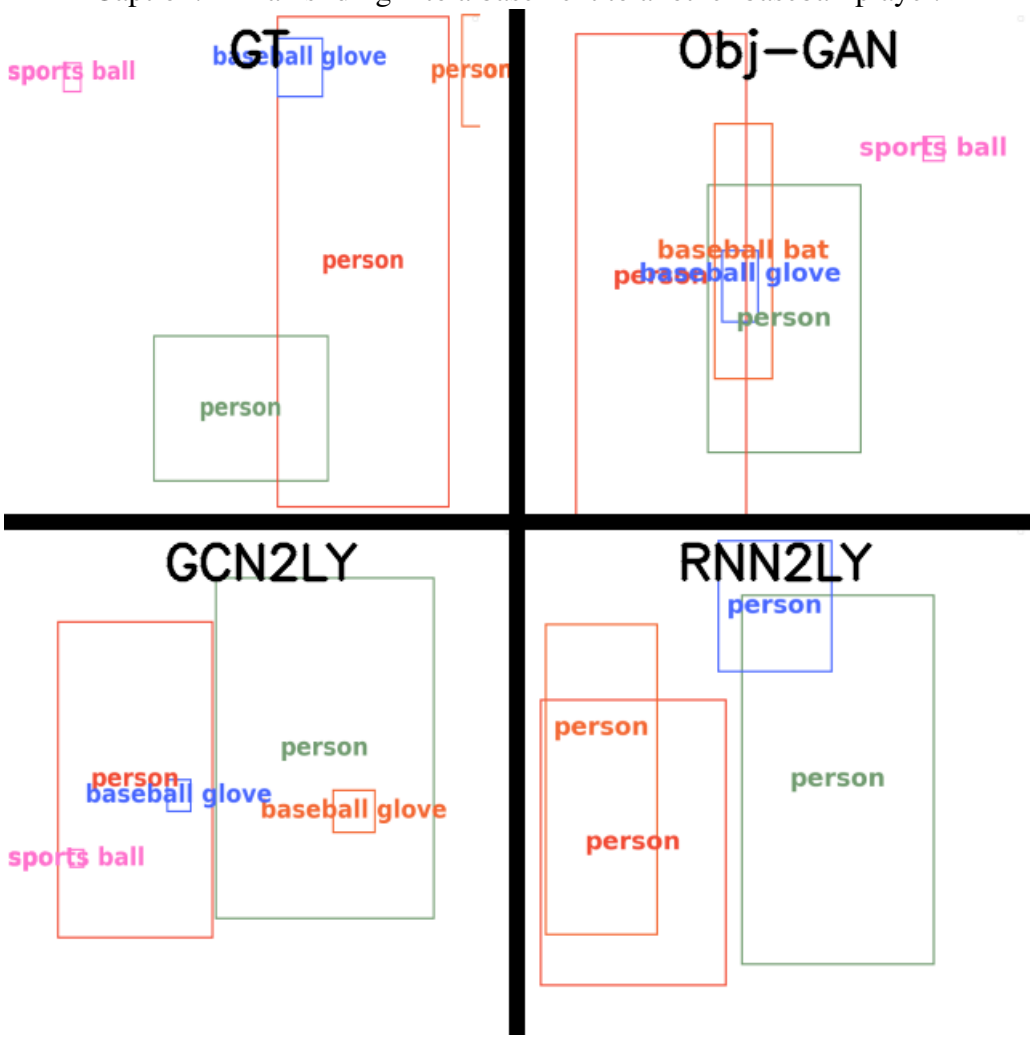


Figure 6.14: A man sliding into a base next to another baseball player. MSCOCO image ID: 515982.

Figure 6.14, shows that Obj-GAN and GCN2LY are able to extract and place correctly the objects related with the caption. These two systems, produce complex layouts maintaining correctly the sizes between objects like the sports ball being much more smaller than a person but similar compared to a baseball glove. Moreover, the baseball glove and baseball bat objects are placed close/overlapping the person as expected from a real layout. Unfortunately, RNN2LY only produces persons, which makes the layout not realistic at all.

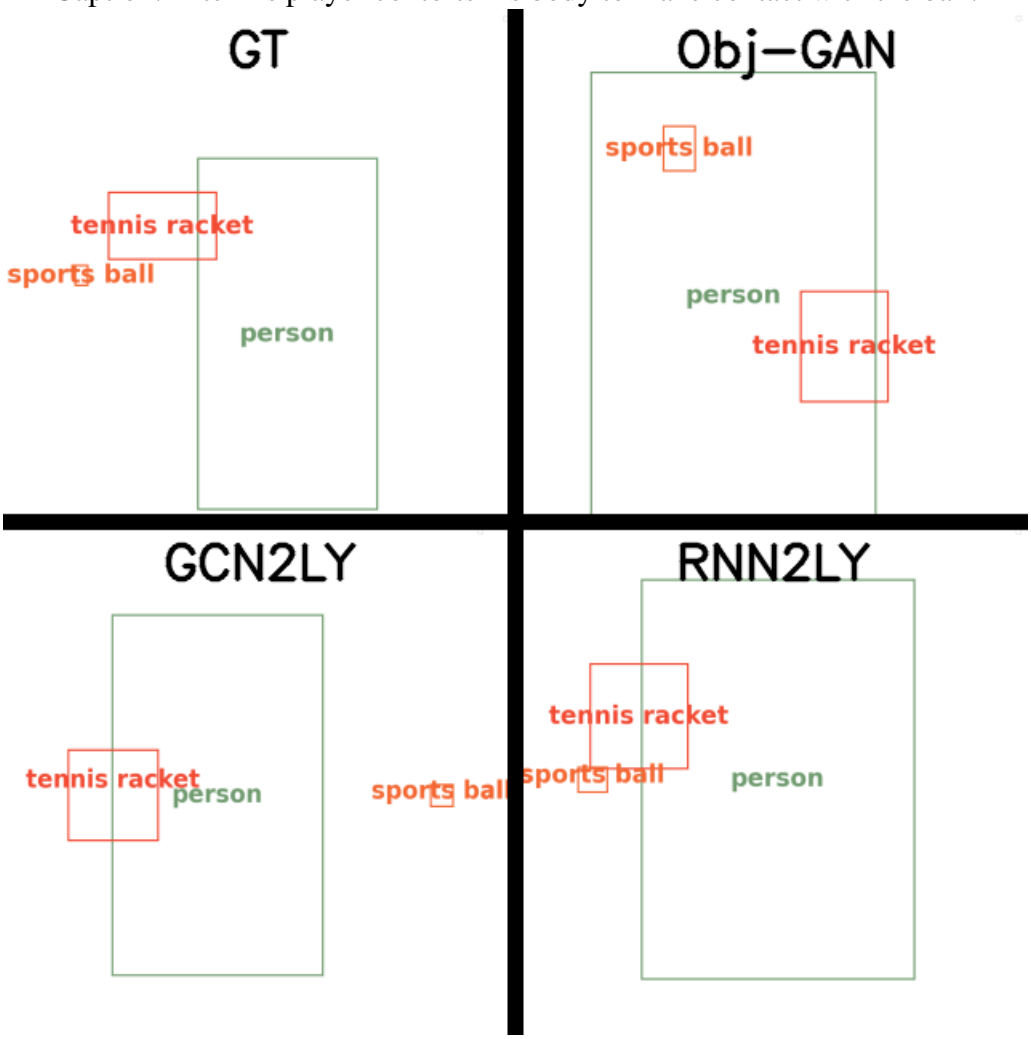


Figure 6.15: A tennis player contorts his body to make contact with the ball. MSCOCO image ID: 520478.

Figure 6.15, shows how the three systems are able to obtain exactly the same objects that appear in the ground truth. Furthermore, the relative sizes of the objects are maintained, such as the tennis racket being bigger than the sports ball but smaller than a person. The three systems, place the tennis racket in a position close to the person's hands, which makes sense with the text description.

Caption: A group of people sit around a table.

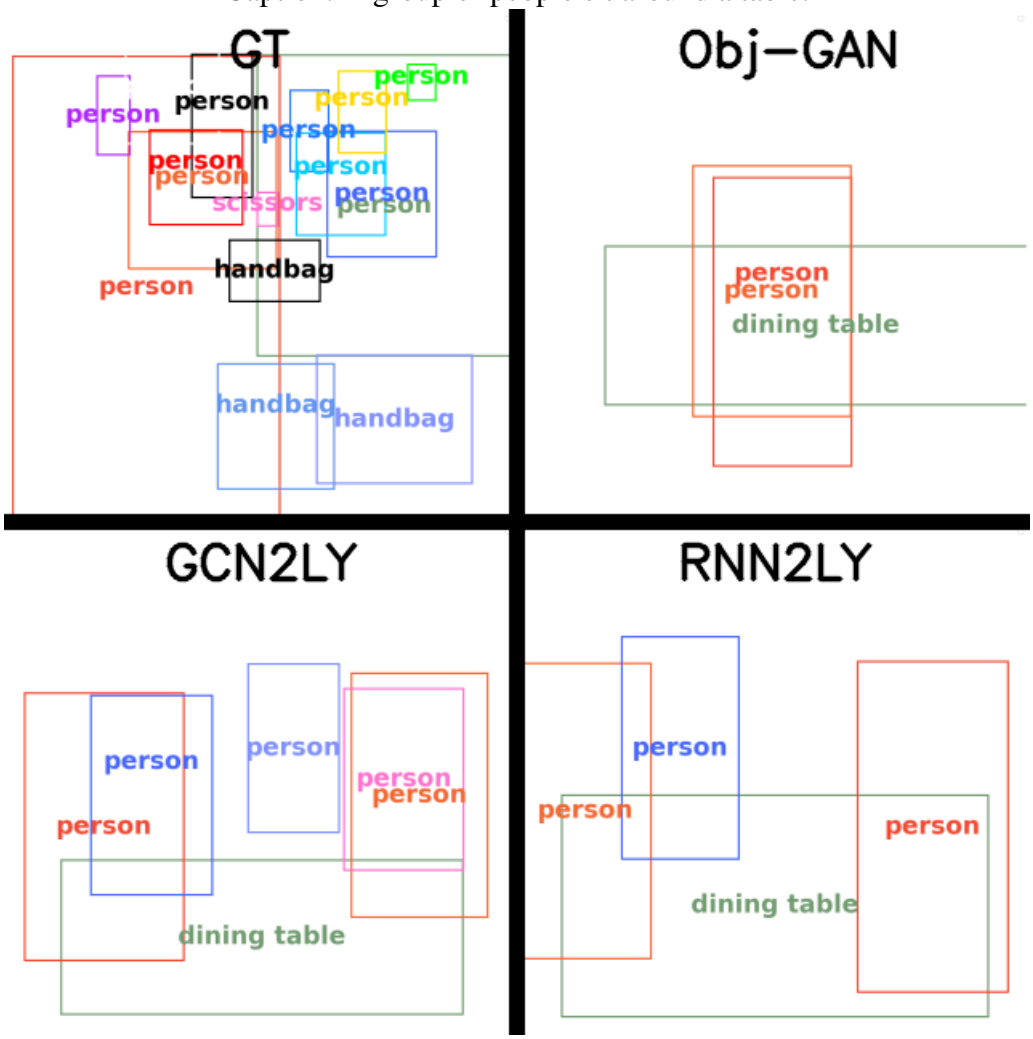


Figure 6.16: A group of people sit around a table. MSCOCO image ID: 10986.

Figure 6.16, shows a ground truth layout with many objects that are not placed according to the text description. Moreover, the word "table" that appears in the caption does not have a bounding box in the ground truth. The three systems, as expected, are able to extract dining table and person objects with correct relative sizes from the caption. Nevertheless, Obj-GAN is unable to place correctly the persons as they appear in the middle, but GCN2LY and RNN2LY are able to understand the concept of "around" as they place the persons around the table.

Caption: A child flies a kite with another child onlooking.

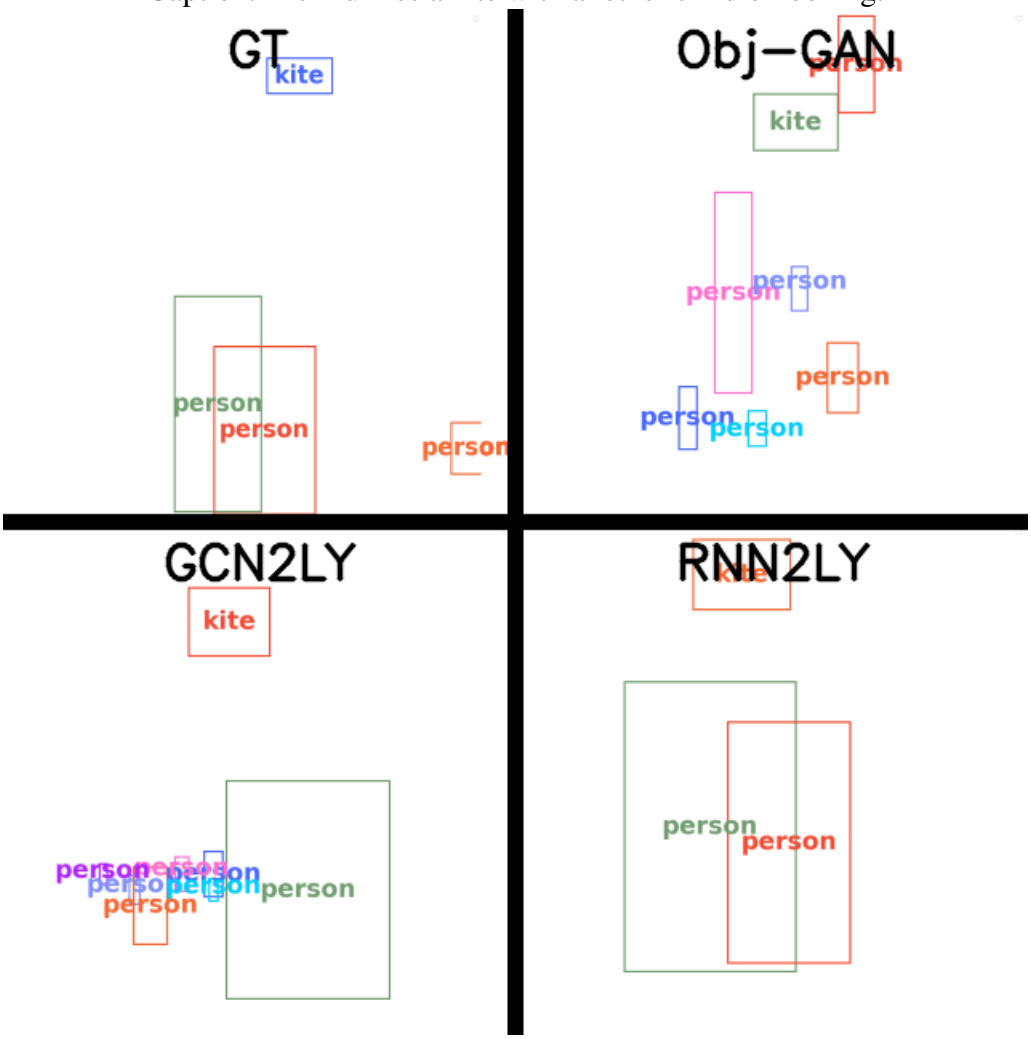


Figure 6.17: A child flies a kite with another child onlooking. MSCOCO image ID: 12543.

Figure 6.17, shows how RNN2LY is able to obtain two persons and a kite as mentioned in the text description. Moreover, the kite is specified as "flying", and therefore it appears in the upper part of the layout. Obj-GAN and GCN2LY, are able to place correctly the kite, nevertheless, too many persons are extracted making the layout not realistic.

Caption: A dog looking up in at a frisbee.

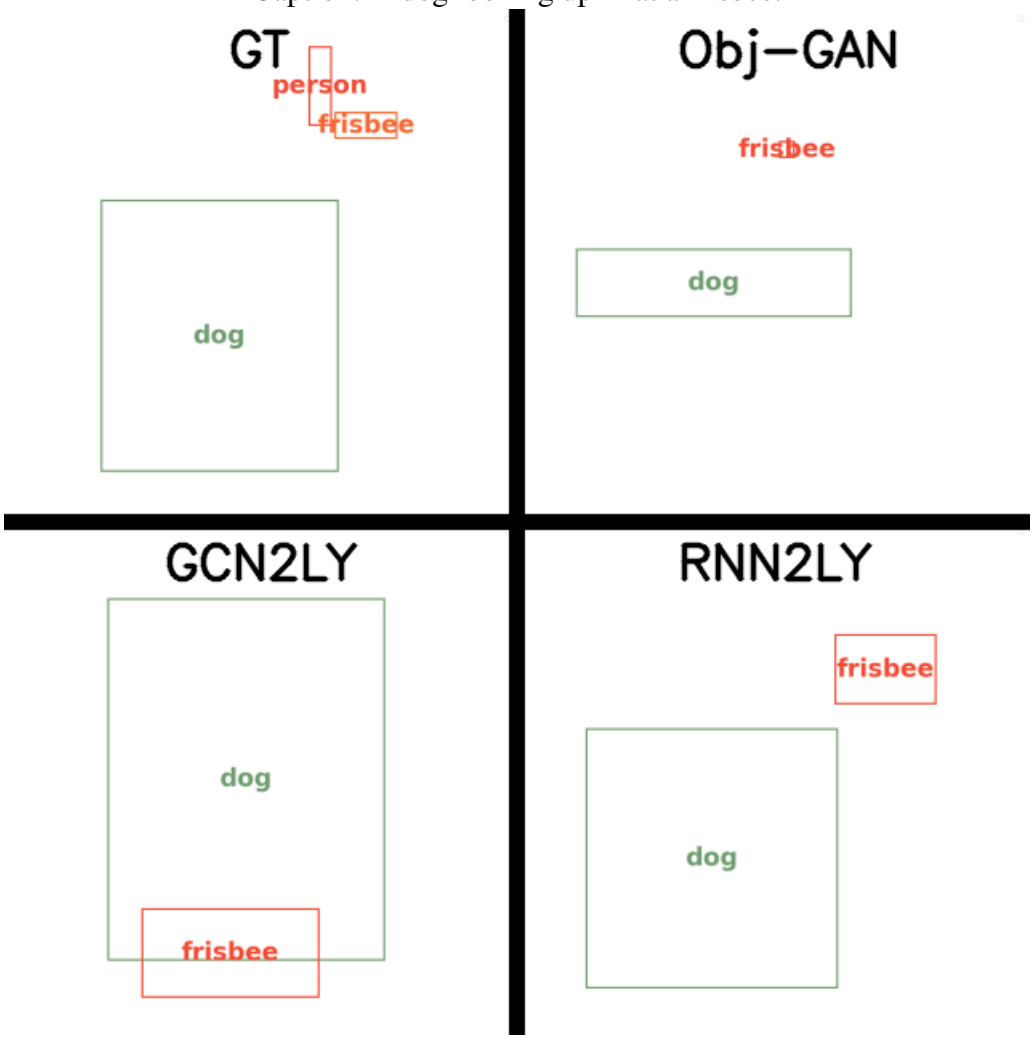


Figure 6.18: A dog looking up in at a frisbee. MSCOCO image ID: 79407.

Figure 6.18, shows how RNN2LY and Obj-GAN are able to understand the relation "looking up" that appears in the caption, as the dog is placed below the frisbee. Nevertheless, the sizes of the objects that appear in RNN2LY are better than Obj-GAN, because in the latter the frisbee is too small. In the case of GCN2LY, it is able to extract the objects correctly, but is not able to understand that the frisbee should be in the upper part of the picture.

These examples clearly show that RNN2LY and GCN2LY are better than Obj-GAN in building complex layouts. The systems developed are able to obtain more objects, maintain the relative sizes of them taking into consideration their interactions, and understand the relations that appear in the text description to properly place the objects.

CHAPTER 7

Conclusions and future work

In this chapter, the conclusions of the project are provided and directions for future work are proposed.

7.1 Conclusions

In this project, two methods for inferring the image layout from a given text description are developed. The systems developed use AMR graphs to see if a structured version of the text contributes to represent better the relations between objects compared to the traditional latent vector representation based on RNNs. In these graphs, nodes are objects to be drawn and edges are the relations between objects. In both systems, graphs are processed using a graph convolutional neural network, which propagates information along the edges in order to learn the relations between objects.

The first system, SG2BB, shows that using directly the node embeddings to obtain the bounding boxes is not enough as it places all of them in the middle of the layout, which in most cases is incoherent with the given description. Moreover, the system may generate objects that do not have a representative meaning due to graphs having nodes with words like “play-01”. In addition, using heuristics to match ground truth objects with graph nodes leads to an inaccurate matching due to not having enough different ground truth object categories in MSCOCO dataset.

The second system, GCN2LY, solves the problems that SG2BB has. The objects are now generated sequentially within a given set of objects categories, which avoids having objects with no meaning as it happened in SG2BB. Bounding boxes are generated using a probability distribution based on a grid system, which allows to obtain the center of the bounding box and then use it in an MLP to obtain the width and height. This allows to generate layouts that are coherent with the text description.

During the development of this project, we saw that there is a necessity to use new metrics to ensure that the generated layouts are coherent with the text description that describes it, therefore, a new set of metrics are proposed that better measure the spatial composition of scenes: relative spatial categorical position, aspect ratio, class matching, and relative scale difference.

The metrics show that GCN2LY is better in all of them compared to the current state-of-the-art system Obj-GAN. Moreover, a visual comparison between the layouts of both systems shows a better spatial relationship between the objects in the scene, which makes it more coherent with the text description.

Unfortunately, the use of a structured version of the text does not incur in any improvement of the generated layout compared to using plain text as seen with RNN2LY system, which modifies GCN2LY system by changing the GCNN by a RNN. Nevertheless, the difference between these two systems is minimal, as the metrics and visual layout are very similar, therefore, we think that a deep research in GCNNs and graph representations could lead to an improvement due to the limitations that RNNs have.

7.2 Future work

This project covers the basic scope of the text-to-layout task which can be continued in the following research lines:

- **Dataset:** The systems have been trained using MSCOCO dataset which offers 80 different object categories. Increasing the number of object categories either using COCO stuff or a completely different dataset could lead to a more meaningful layout that can be used to draw more complex layouts.
- **Graph representation:** In this project, we make use of AMR graphs. Searching or developing a new graph representation to build a better structured version of the text that is more suitable for text-to-layout task could lead to improve GCN2LY results.

- **GCNN architectures:** In this project, SG2BB graph convolutional neural network is used to process the graphs. Nevertheless, this architecture is not the only one available, therefore, a different type of GCNN could lead to an improvement of the results.

- **Transformer encoder:** In this project a GCNN is used to propagate information along the edges in order to learn the relations between objects and obtain a representation. Nowadays, transformers [Vaswani et al., 2017] are used in state-of-the-art systems, therefore, using a pretrained transformer like BERT or training a new one as an encoder could lead to a more complex and rich representation of the text that could improve the generated layout by the decoder.

- **Transformer decoder:** In this project, an LSTM is used to generate the layout of the text description. In the same way as the previous point, a transformer could also be used as a decoder in order to generate a more complex layout and improve the spatial relationship between the objects in the scene.

- **Text-to-image:** Text-to-image is the task of generating pictures from text descriptions. As mentioned in the introduction, a three-step process can be used to generate pictures and this project could be the first step of it.

Appendix

CHAPTER A

Appendix

A.1 Project objectives report

In this chapter, the objectives of the project are defined. The project definition covers an overall description of the project, the concrete goals of the project, and the planning and the methodology to achieve those goals. Finally, a list of identified risks that can compromise the project is given.

A.1.1 Project description and goals

The main objectives of this project are to develop a new architecture that generates a layout from a given text description, evaluate it thoroughly using different metrics, and compare it with other related systems. To accomplish the previous objectives the work can be divided into smaller tasks:

- Study the literature and understand the problem.
- Evaluate state-of-the-art implementations.
- Define and evaluate the dataset.
- Evaluate different ways to represent the text.

- Define the architecture.
- Define the metrics.
- Systems experiments and comparison.

The previous tasks will give a deep understating of how the research world works for real-life problems.

A.1.2 Project planning

WBS diagram

A Work Breakdown Structure (WBS) is used to outline the work that needs to be done for this project and is shown in Figure A.1. The project time estimates for each work unit can be seen in Table A.1.

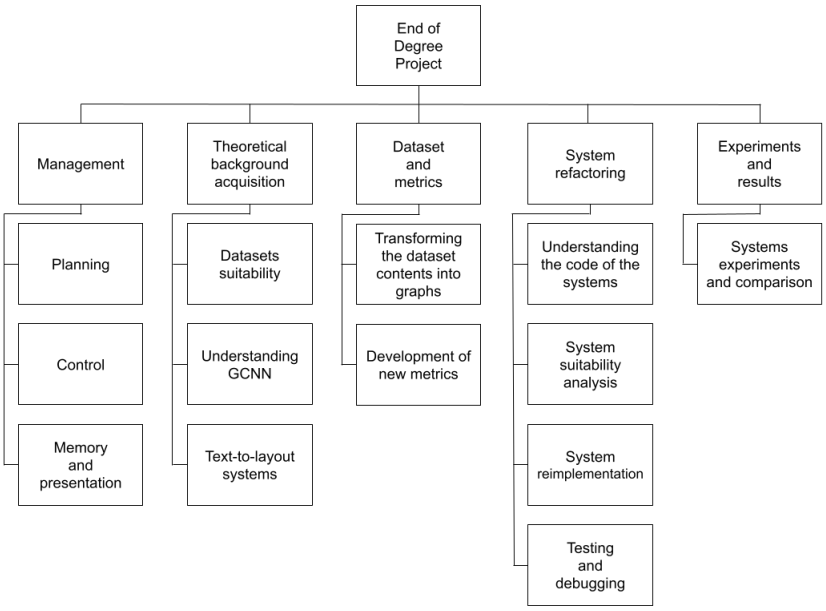


Figure A.1: Work Breakdown Structure of the project.

Work units

In this section, a summary of each work unit and the estimation of the time that will be spent on each one are going to be given. Because of the difficulty of this project, some of these tasks may require more time than expected.

Planning

The work related to organizing the project that includes: what the project is about, what are the goals, what are the tasks and when they need to be delivered, what milestones have to be achieved and the deadline for those milestones, and what are the risks of this project. The deliverable of this task is this report.

Control

Keeping the project focused, achieving the goals proposed according to the established schedule, and solving any problem that could affect the project. This work is performed in control meetings every week.

Memory and presentation

Writing the memory of the project with the work that has been made and the public defense of it which will require to prepare materials for the presentation.

Datasets suitability

Search and study of datasets suitable for the project.

Understanding GCNN

Understanding an almost unused architecture in deep learning research. This will require understanding well papers and different implementation approaches as there is huge lack of information about it.

Text-to-layout systems

Investigate text-to-layout systems. This task requires to read papers about text-to-image, as text-to-layout systems are almost nonexistent.

Transforming the dataset contents into graphs

Research different ways to transform text to graphs focusing on rules-based systems and deep learning state-of-the-art systems with pretrained models.

Development of new metrics

Analyze the suitability of developing new metrics for text-to-layout systems and develop them if convenient.

Understanding the code of the systems

Understanding the code of state-of-the-art systems. This will include understanding how GCNNs are implemented and the potential improvements that can be applied to the systems in order to improve their performance.

System suitability analysis

Analyze if the systems are suitable for our project requirements and what changes need to be done in order to fit our ideas.

System reimplementation

Reimplementation of systems that are old, not trained, or not working properly that may affect our timeline.

Testing and debugging

Testing and debugging the reimplementation or modifications of the systems in order to ensure that they work properly.

Systems experiments and comparison

Use the newly proposed metrics in the state-of-the-art and developed systems to compare the improvements. This task will also include a visual comparison between all of them to ensure that the new metrics work as intended.

Work-unit	**Time estimate (hours)**
Management	**150**
Planning	5
Control	45
Memory and presentation	100
Theoretical background acquisition	**100**
Datasets suitability	10
Understanding GCNN	30
Text-to-layout systems	60
Dataset and metrics	**30**
Transforming the dataset contents into graphs	10
Development of new metrics	20
System refactoring	**200**
Understanding the code of the systems	20
System suitability analysis	5
System reimplementation	150
Testing and debugging	25
Experiments and results	**20**
Systems experiments and comparison	20
Total	**500**

Table A.1: Time estimates for each work unit.

Gantt chart

A Gantt chart is used to illustrate the project schedule. This chart lists the tasks to be performed on the vertical axis and the time intervals on the horizontal axis. The Gantt chart is shown in Figure A.2.

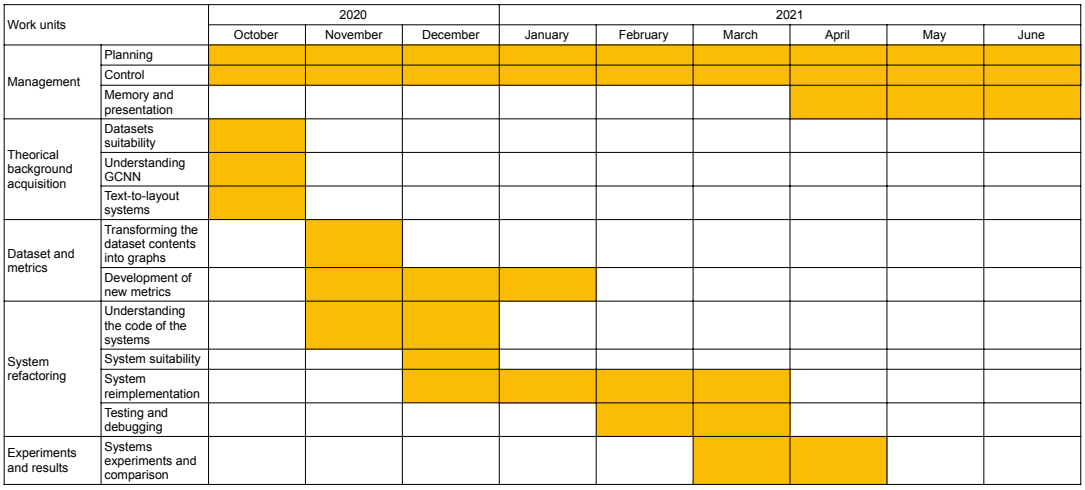


Figure A.2: Gantt chart of the project.

Milestones

Table A.2 shows the deadline dates for the deliverables.

Deliverable	**Date**
Implementation	20/06/2021
Memory	20/06/2021
Presentation	28/06/2021 - 09/07/2021

Table A.2: Deliverables and their deadlines.

A.1.3 Methodology

This undergraduate thesis is carried out as a project of the IXA natural language processing research group. The student receives support from two IXA instructors (Gorka Azkune Galparsoro and Oier López de Lacalle Lecuona). The student is allowed to use hardware resources from the IXA research group in the form of server nodes with CPU and GPU capabilities.

Meetings

Regular meetings are arranged in a fixed slot every week with the two IXA instructors. These meetings are held, either in the faculty or remotely using Google Hangouts. These meetings' objective is to control the progress of the project and talk about problems and their possible solutions. Meetings outside the fixed time are scheduled to attend specific doubts.

Work place

The student will work from home in a relaxed environment and good internet connection.

A.1.4 Risks

Given the size and scope of the project, there may be some uncertainties that will put the project at risk. The following list describes some of the risks identified:

- **COVID-19 situation:** The uncertainty of the current situation may incur some unexpected situations, such as the impossibility of holding the meetings in the faculty and the student or instructors getting infected and therefore, the impossibility to work and hold meetings for a while.
- **Compute capabilities:** Deep learning requires high amounts of computing and memory resources. The student is given the possibility to use the IXA research group's shared resources in case the online free resources are not enough. These shared resources may be in high demand for a given period, therefore, it is difficult to know when they will be free.

- **Research factors:** The project has a research component that adds uncertainty to it due to the bare use of GCNNs in the research community and the small number of papers related to text-to-layout task.

Acknowledgments

This project has been partially supported by the Basque Government through the Ikasiker grants program.

Bibliography

[Akbik et al., 2019] Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019). *FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP*. Association for Computational Linguistics.

[Andreas et al., 2016] Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Learning to compose neural networks for question answering. *CoRR*.

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR*.

[Cai and Lam, 2020] Cai, D. and Lam, W. (2020). AMR Parsing via Graph-Sequence Iterative Inference. *ACL*.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*.

[Everingham et al., 2014] Everingham, M., Eslami, S. M. A., Gool, L. V., Williams, C. K. I., Winn, J., and Zisserman, A. (2014). The PASCAL Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *NeurIPS*.

[Gori et al., 2005] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. *IEEE International Joint Conference on Neural Networks*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.

[Hong et al., 2018] Hong, S., Yang, D., Choi, J., and Lee., H. (2018). Inferring Semantic Layout for Hierarchical Text-to-Image Synthesis. *CVPR*.

[Józefowicz et al., 2016] Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *CoRR*.

[Justin et al., 2018] Justin, J., Gupta, A., and Li, F.-F. (2018). Image Generation from Scene Graphs. *CPRR*.

[Kingsbury and Palmer, 2002] Kingsbury, P. and Palmer, M. (2002). From Treebank to PropBank. *In Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC)*.

[Kuhn., 1955] Kuhn., H. W. (1955). The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*.

[Li et al., 2019a] Li, B., Qi, X., Lukasiewicz, T., and Torr, P. H. S. (2019a). Controllable Text-to-Image Generation. *NeurIPS*.

[Li et al., 2019b] Li, W., Zhang, P., Zhang, L., Huang, Q., Xiaodong He, S. L., and Gao, J. (2019b). Object-driven Text-to-Image Synthesis via Adversarial Training. *CVPR*.

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014). Microsoft COCO: Common Objects in Context. *CoRR*.

[Mahmood and Len, 2017] Mahmood, Y.-A. and Len, H. (2017). Text summarization using unsupervised deep learning. *Expert Systems with Applications*.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *ICLR*.

[Mikolov et al., 2013b] Mikolov, T., Corrado, G., Chen, K., and Dean, J. (2013b). Efficient estimation of word representations in vector space. *CoRR*.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). *GloVe: Global Vectors for Word Representation*. Association for Computational Linguistics.

[Powers, 2019] Powers, D. M. W. (2019). Evaluation: From precision, recall and f-factor to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*.

[Ramesh et al., 2021] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. *arXiv:2102.12092*.

[Reed et al., 2016] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative Adversarial Text to Image Synthesis. *ICML*.

[Robbins, 2007] Robbins, H. (2007). A stochastic approximation method. *Annals of Mathematical Statistics*.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*.

[Scarselli et al., 2009] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

[Shen et al., 2017] Shen, Y., Huang, P.-S., Gao, J., and Chen, W. (2017). *ReasoNet: Learning to Stop Reading in Machine Comprehension*. Association for Computing Machinery.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *CoRR*.

[Tan et al., 2019] Tan, F., Feng, S., and Ordonez, V. (2019). Text2scene: Generating compositional scenes from textual descriptions.

[Tripathi et al., 2019] Tripathi, S., Bhiwandiwalla, A., Bastidas, A., and Tang, H. (2019). Using scene graph context to improve image generation. *CoRR*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.Ñ., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*.

[Wu et al., 2019a] Wu, H., Mao, J., Zhang, Y., Jiang, Y., Li, L., Sun, W., and Ma, W.-Y. (2019a). Unified Visual-Semantic Embeddings: Bridging Vision and Language With Structured Meaning Representations. *CVPR*.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*.

[Wu et al., 2019b] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019b). A Comprehensive Survey on Graph Neural Networks. *CoRR*.

[Xu et al., 2017] Xu, T., Zhang, P., Huang, Q., Zhang, H., Gan, Z., Huang, X., and He, X. (2017). AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. *CoRR*.

[Zhang et al., 2017] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. (2017). StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. *ICCV*.

[Zhu et al., 2019] Zhu, M., Pan, P., Chen, W., and Yang, Y. (2019). DM-GAN: Dynamic Memory Generative Adversarial Networks for Text-To-Image Synthesis. *CVPR*.