Image generation from scene graphs

Kalmár Gábor, Kostyál Domonkos Marcell and Tárnok Márton

Abstract

This is a university project for the subject Deep Learning in Practice with Python and LUA (VITMAV45) [2]. Our goal was to recreate and implement the publication of Johnson, Gupta, Fei-Fei (2018)[1]. Our model uses graph convolution to process input graphs, computes a scene layout by predicting bounding boxes and segmentation masks for objects, and converts the layout to an image with a cascaded refinement network. We used the COCO Dataset[3] for training, but due to lack of computation resource, we decided to use a smaller set of objects, and we dispensed with implementation of the image pair of discriminators.

1. Introduction

The goal of the project was to gain a deeper understanding of graph convolutional networks and image generation using text. A complex sentence can often be more explicitly represented as a scene graph of objects and their relationships. Scene graphs are a powerful structured representation for both images and language.

2. COCO Dataset

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features.

From almost 20 GB of images, we sorted 10000 pieces for training, validating and testing. We will use a 10% testing and a 10% validation split.
We cropped and resized the images to 64x64 and sorted out the images with the maximum of 4 objects to reduce computational time. We selected only a smaller group of objects (person, giraffe), so the model can only generate images from scene graph containing these elements.

3. Input Data

We started to work with Jupyter Notebook, at our first try we loaded the Coco annotations file, that contains the objects and their relations, the coordinates of the bounding boxes and masks. We converted all this data into a tensor, this will be the input for the training. For visualization we created outputs in JSON format.

The input to our model is a scene graph describing objects and relationships between objects. Given a set of object categories C and a set of relationship categories R, a scene graph is a tuple (O, E) where O = {o1, . . . , on} is a set of objects with each oi ∈ C, and E ⊆ O × R × O is a set of directed edges of the form (oi , r, oj) where oi , oj ∈ O and r ∈ R.

For example, the scene graph output for an image will look like this:


```
{"ID": 522418, "relationships": [["person", "above",
"knife"], ["knife", "below", "person"], ["cake", "left
of", "person"], ["person", "__in_image__", "__image__"],
["knife", "__in_image__", "__image__"], ["cake",
"__in_image__", "__image__"]]}
```

Fig. 1: Picture and scene graph

they use the the output of the Graph Convolutional Net. Boxnet is a simple MLP, which places the bounding box to the right place of the image. The Mask net is a CNN that predicts masks and fits them into the size of the bounding box. By combining these two, we get the place and mask of the object on the image. By summing all object layouts gives the scene layout

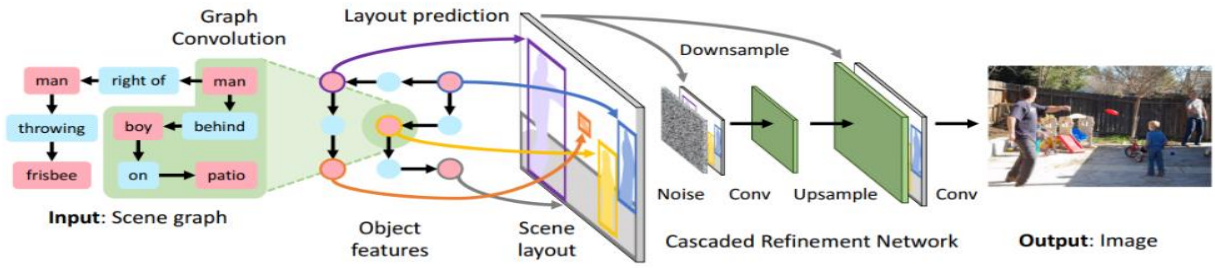

Fig. 2: Overview of model (from the original paper)

4. Method and Pipeline

The model as mentioned above takes as input a **scene graph** describing objects and their relationships. These are fed to a **Graph Convolutional Network** which outputs a grid of vectors. This output is given to the **mask and box nets,** which produce the object layout. From the object layouts a **scene layout** is created. From the scene layout the synthetized image is created, using a **Cascaded Refinement Network**.

4.1. Graph Convolutional Network

This network consists of graph convolutional layers. The output of the network is a grid of vectors, where each output vector is a function of a local neighbourhood of its corresponding input vector, this way the convolution aggregates information across local neighbourhoods of the input. The input contains the objects predicates and edges of the scene graph, and We use ***Embedding network*** to convert objects and predicates into vectors.

4.2. Mask and Box Nets

These two nets are responsible for generating object layouts. They are trained at the same time and

4.3. Cascaded Refinement Network

After generating the scene layout, we synthesize an image that respects the object positions given in the layout. A Cascaded Refinement Network consists of a series of convolutional refinement modules. The first module takes Gaussian noise as input, and the output from the last module is passed to two final convolution layers to produce the output image.

5. Training and improvements

Once the model ran we started experimenting with different paramters.

```
self.vocab = vocab
self.image_size = (64, 64)
self.embedding_dim = 128
self.gconv_dim = 128
self.gconv_hidden_dim = 512
self.gconv_pooling = 'avg'
self.gconv_num_layers = 5
self.refinement_dims = (1024, 512, 256, 128, 64)
self.activation = 'leakyrelu-0.2'
self.mask_size = 16
self.mlp_normalization = 'batch'
self.layout_noise_dim = 32
```

Fig 3.: Parameters of model

This was the first ever picture that the model generated of a giraffe:

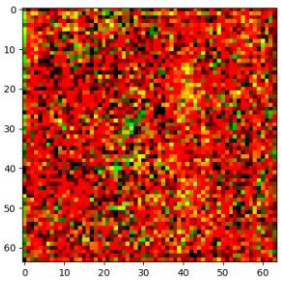

Fig 3.: Our first generated picture

The picute is full of noise, and it is not normalized, but still there can be seen a yellow shape on the right side of the image. This is because at the first few epochs a the original layout (masks and boxes) is placed on the image, because if the mask predictions are not accurate, the refinement network losses are getting falsly huge.

Applying the original masks help the network generate better pictures.

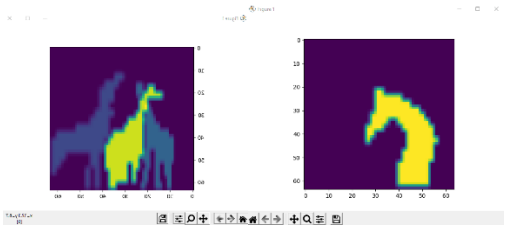

Fig. 4: Scene Layouts of giraffes using the original masks and bounding boxes, that are given to the CRN

After training the model for few epochs we got slightly better results, but the images were oversaturated and noisy.

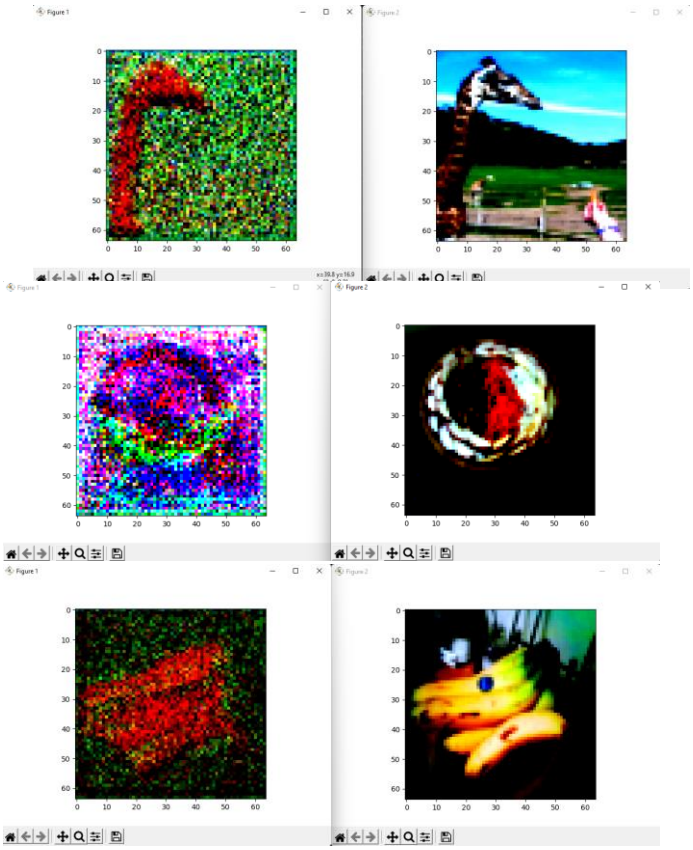

Fig 5.: Noisy, oversaturated pictures

After inverting the standardization we got much smoother results:

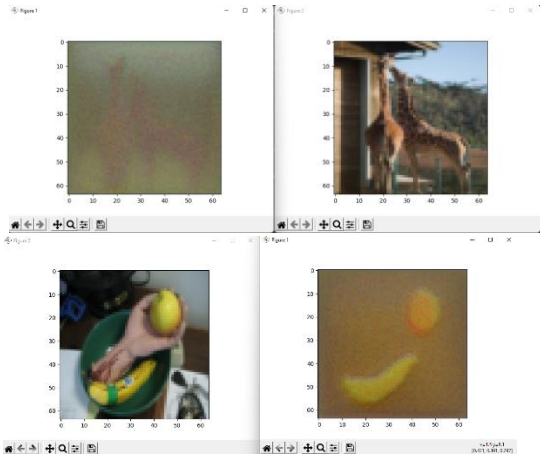

Fig 6: Smoother pictures, the one at the bottom only generates the banana and the apple, because only these two filters were given.

With the parameters tested, we started to train the model for longer time periods.

We have 20755 pictures, which contains 54071 objects, which means 2.61 objects per picture. There are 648 batches in one epoch, each with 32 images.

After 15th epoch we got this result:

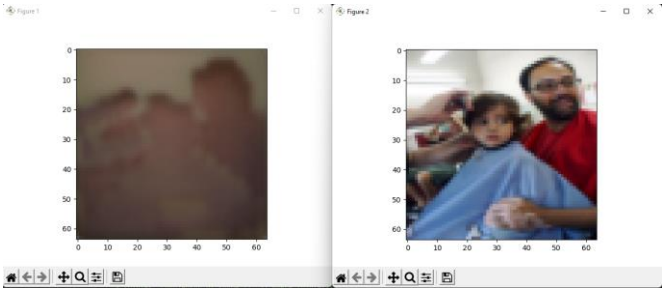

Fig 7.: It is clearly visible that the shape line of the people has a darker colour, and their hairs are also darker

After 30th epoch we got this result:

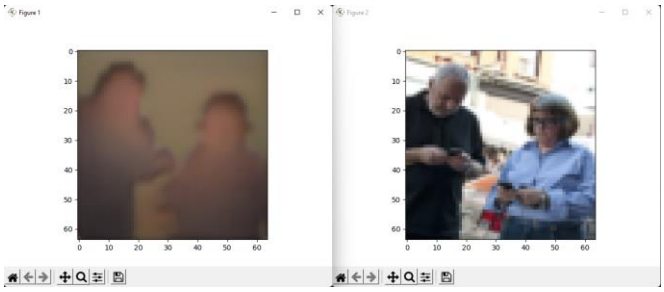

Fig 8.: The pictures have much better tone, their skin and clothes can be differentiated.

After 60th epoch we got this result:

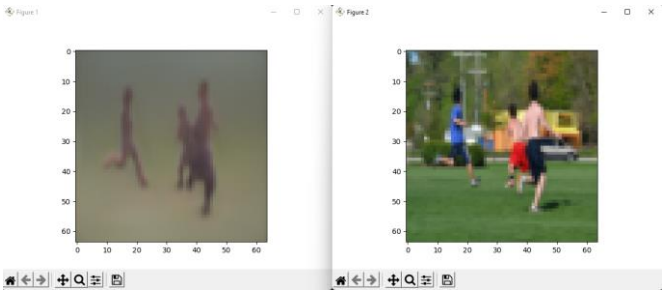

Fig 9.: The people are observed from further distance, the results are worse at this one

We ran 62 epochs because time limitations. After this train the model generated a for the next scene graphs the next pictures:


```
{
  "objects": ["person", "person", "person"],
  "relationships": [
    [0, "left of", 1],
    [2, "right of", 1]
  ]
},
```


Fig 10.: It is visible that there are three person on the generated image


```
{
  "objects": ["giraffe", "giraffe"],
  "relationships": [
    [0, "right of", 1]
  ]
},
```


Fig 11.: It is clearly visible that in the train datasets, the giraffes mostly were in groups of two with crossed necks

6. Workflow, experiences

We started the project by reading the given document several times and because none of us worked on a project like this before, we started reading about graph convolutional networks and synthetic image generation. We split the parts of the project into three different parts, and everyone looked up examples. We organized weekly meetings, where we discussed the improvements, and impletented the parts of project one by one. We

had several difficulties during the semester. The greatest challenge was, that the end result is only visible after the implementation of the whole pipeline, which took more than eight weeks. We also got stuck at ther object layouts, where the mask and boxnets had to be trained paralelly. At the end, we had problems due to the lack of computational, after training for two nights with slightly better result, we decided to change some parameters and train with fewer data.

After all, we believe that the project was a major success, we gained a lot of knowledge, and grow together as a team. We might consider working an projects like this in the future.

7. References

[1] JOHNSON, Justin; GUPTA, Agrim; FEI-FEI, Li. Image generation from scene graphs. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018. p. 1219-1228.Surname A and Surname B 2009 Journal Name

[2] BME VIK - Deep Learning a gyakorlatban Python és LUA alapon

[3] http://images.cocodataset.org/zips/train2017.zip

[4] http://images.cocodataset.org/annotations/annotations_trainval2017.zip

[5] Our repo: 5aola/ImGen: Image generation from scene graphs (github.com)