Image Synthesis From Hand Drawn Reconfigurable Layouts

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Ganji Akhil Kumar
[Roll No: CS-2015]

under the guidance of

Dr. Ujjwal Bhattacharyya.
Associate Professor
Computer Vision & Pattern Recognition Unit

Indian Statistical Institute
Kolkata-700108, India

July 2022

CERTIFICATE

This is to certify that the dissertation entitled **"Image synthesis from Hand Drawn Reconfigurable Layouts"** submitted by **Ganji Akhil Kumar** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

[illegible]

Dr. Ujjwal Bhattacharyya
Associate Professor,
Computer Vision & Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.
Date : 05/07/2022.

Acknowledgments

I would like to express my profound gratitude to my supervisor, Dr. Ujjwal Bhattacharyya, for all of his guidance and assistance. I also want to express my sincere gratitude to all the researchers who are working in the Visoin Lab of the Computer Vision and Pattern Recognition Unit, in my institute, the Indian Statistical Institute, Kolkata,

I am thankful to the faculty and staff members of the Department of Computer Science, for allowing me to avail the facilities of the department

I would like to thank all of my friends for their valuable discussions and support.

G. Akhil Kumar

Ganji Akhil Kumar
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

Image synthesis is a significant computer vision problem with numerous applications. With the rise of Generative Adversarial Networks, there has been a significant advancement in this area (GANs). Recent times have seen a rise in interest towards conditional image generation from layout. To create useful applications with a user-friendly interface, taking control of the image generating process is essential.

The focus is to study generative models for generating almost real images from the spatial layout in which bounding boxes of objects and their categories are configured in an image lattice, and style codes (i.e., latent vector encoding structural variation). The study of intuitive paradigm for the problem, layout to mask to image is done. TO connect the dap between input layout and synthesized images, layout to mask component major role as it deeply interacts with the generator network. A GAN is built for layout to mask to image synthesis with style control and layout control at both object level and image level. The controllablility is realised by ISLA Norm (Instance Sensitive and Layout Aware Normalization) scheme. We create and experiment on a the challenging Visual Genome dataset.

Contents

Chapter 1

Introduction

The progress of image synthesis research and applications was severely influenced by the advent of Generative Adversarial Networks (GANs). With current methods, it can be quite difficult to tell high resolution face images from low resolution, grayscale face images apart from real images. Even though unconditional image synthesis is intriguing, most real-world applications require an interface that lets users tell the model what to generate. In recent years, conditional generative approaches have incorporated segmentation masks, class labels, images, speech, text, layout, and combinations of these to obtain control over the image synthesis process. However, the majority of these methods use ”one-shot” picture generators, which restrict changing some characteristics of the generated image.

Although iterative image manipulation has made significant progress, researchers have not yet researched into ways to have more control over the image generation of complex scenes with many interacting objects. The system must be able to iteratively and interactively update the image in order to let the user create a scene that reflects what he or she has in mind.By permitting customizable spatial layout and object styles, a recent approach by Sun and Wu makes a significant advancement towards this objective. Each object in their method has a latent style code that is used to generate new images and is sampled from a normal distribution.

1.1 Image Synthesis

Image synthesis (generation) is the task of generating new images from an existing dataset. They are various types of image synthesis tasks, of which two are :

1. Unconditional image synthesis refers to generating samples unconditionally from the dataset, i.e. $p(y)$

2. Conditional image synthesis (subtask) refers to synthesising samples conditionally from the dataset, based on a label, i.e. $.p(y/x)$

1.2 Visual Genome Dataset

Visual Genome Dataset is a huge collection of real world images, each equipped with annotations of various regions in the image. The annotations include a plain text description of the region (often comprised of sentence parts or short sentences, such as "a red ball in the air") as well as several other forms of properly gathered information (attributes, objects, relationships, scene graphs, region graphs, and question-answer pairs). Over 108K images in the Visual Genome dataset have an average of 35 objects, 21 pairwise relationships and 26 attributes between objects.

1.3 Problem Statement

The interest is to be able to achieve controllable image synthesis from reconfigurable layouts. By reconfigurable and controllable, it means a generative model is capable of

1. *Layout Control* – The model is adaptable to changes in spatial layouts within a certain layout, as well as the style codes associated to all those changes.
2. *Style Control* – At both the image and object levels, the model maintains and preserves the underlying one to many mapping from a given layout to numerous plausible images with sufficiently different structural and appearance styles.

Let X be an image lattice (for example, 128 $\times$ 128) and by I, an image outlined on the lattice. Let L = $(l_i, bbox_i)_{i=1}^m$ be a layout with m bounding boxes that are labelled, where a label $i \in C$ (e.g., C is no. of objects like $|C| = 179$ in the Visual genome dataset), and a bounding box $bbox_i \subseteq X$. Different bounding boxes may overlap and resulting in an arbitrary partial order of occlusion.

Let z_{obj_i} be the latent code that controls the object instance style and z_{img} be the latent code that controls the image style for $(l_i, bbox_i)$. Under the i.i.d. setting, the latent codes are often randomly sampled from the standard multivariate Gaussian distribution, N(0, 1). Denote by $Z_{obj} = z_{obj_i}{}_{i=1}^m$ the set of object instance style latent codes. The goal of learning an image synthesis from layout and style is to translate a given input layout (L, z_{img}, Z_{obj}) to a synthesised image, I^{syn},

$$I^{syn} = G(L, z_{img}, Z_{obj}; \Theta_G),$$

where Θ_G represents the generation function parameters. In general, underlying conditional data distribution $p(I|L,\ z_{img}, Z_{obj}\ ;\ \Theta_G)$ of high dimensional space is anticipated to be captured by a generator network G. Although simple to use it for generating synthetic images, the generator G has a challenging inference phase that must be completed in order to estimate the model's parameters: computing the latent codes for a real image I.

Chapter 2

Previous Work

Generative models such as Variational Autoencoders (VAEs) Autoregressive models and Generative Adversarial Networks (GANs) have been studied widely in the recent years. VAEs simultaneously train an encoder and a decoder, with the former converting images into a latent distribution and the latter creating images based on the distribution. PixelCNNs and PixelRNNs are examples of image generation for autoregressive models that create images pixel by pixel based on conditional distribution over pixels. GANs are capable of synthesising realistic and high-resolution images under diverse conditions, including unconditional and conditional tasks. A GAN typically comprises of a Generator, which produces realistic fake images from input (such as random noise), and a Discriminator, which distinguishes synthesized images from real ones. The model used, which is based on GANs and aims to generate images conditioned on coarse semantic layouts, was developed.

2.1 Conditional Image Synthesis

In conditional image synthesis, additional data, such as class information, the input image, a text description, and scene graphs, are used as inputs. Numerous approaches have been studied to determine how to feed conditional information to a GAN model. First, conditional information is encoded into a vector representation in each technique. Different approaches employ the encoded condition vector in different ways. In Generative adversarial text to image synthesis and Conditional image synthesis with auxiliary classifier gans, the encoded condition vector and a sampled latent vector are concatenated as the input to the generator network. In StackGAN and Generative adversarial text to image synthesis, the encoded condition vector is utilized by the discriminator by simply concatenating with the input or intermediate feature maps. The inner product between the discriminator's features and the encoded condition vector is often used in projection-based approaches in cGANs with projection discriminator to effectively increase the quality of class conditional image generation. In Large scale

GAN training by A. brock et. al, Semantic image synthesis by T.Park et. al, learned representation for artistic style by V. Duomulin et.al, the conditional BatchNorm is constructed by using the encoded condition vector to govern the re-scaling and re-shifting parameters in the BatchNorm layers. An annotated semantic label map is used by GauGANs to further develop spatially adaptive re-scaling and re-shifting parameters for BatchNorm. The proposed ISLA-Norm in the earlier LostGAN-V1 is a concurrent effort with the feature normalisation approach in GauGANs, but without the use of annotated semantic label mappings. From coarse layout information, it learns the layout to mask mapping. Additionally, the projection-based techniques for exploiting conditional information in the discriminator are adopted by the suggested LostGANs.

2.2 Image Synthesis from Layout

In the most recent literature, image synthesis from layout has been explored and shown to be a challenging task. Layout2image by B. Zhao et al. is the first study on the layout to image task, and it uses a variational autoencoders based network together with long-short term memory (LSTM) for object feature fusion. In Pastegan, authors provide an external memory bank that contains objects that were cropped from real images during training and are then retrieved and pasted to synthesize images from layouts with a resolution of 64x64. In Image generation from scene graphs by J.Johnson et. al, Object-driven text to image synthesis by W. Li et.al, layout and object information are utilised in text to image synthesis and scene-graph to image synthesis to generate scene images from given masks by matching the shape, context and parts to a library that is maintained. Locations of multiple objects in text to image synthesis are controlled by including an additional object pathway in both the generator and discriminator, according to T. Hinz et al papers "Generating multiple objects at spatially distinct locations" and "Semantic object accuracy for generative text to image synthesis." The two processes involved in image synthesis are creating the semantic layout (bounding boxes, class label, and semantic mask) from a text description or a scene graph, and then synthesising images conditioned on the predicted semantic layout and text description.

Chapter 3

Related Work

3.1 Class-Conditional Image Synthesis

Probably the most direct way to gain control on what image to generate is to generate images with a class label. Initial approaches condition the generator by concatenating the noise vector with encoded label. Recent approaches have considerably improved the resolution, image quality, and diversity of generative models. However, they have two major drawbacks that restrict their practical application: they are based on single-object datasets, and they do not allow the reconfiguration of individual aspects of the image to be generated.

3.2 Layout to Image

The direct layout to image problem was initially studied in Layout2Im using a VAE-based approach that could generate a variety of 64x64 pixel images by breaking down the representation of each object into a specified label and an unspecified (sampled) appearance vector. Using a reconfigurable layout, LostGAN improves control over individual objects while maintaining the integrity of existing objects in the output image. This is accomplished by giving each object an individual latent style code, whereby one code for the whole image allows the generation of diverse images from the same layout when the object codes are fixed.

3.3 Scene-Graph to Image

Scene graphs use a graph structure to represent a scene of multiple objects, with the objects being encoded as nodes and the relationships between them as edges. Scene graphs have recently been used in a multiple image generation approaches due to their practical and flexible structure. Typically, a scene layout with segmentation masks and bounding boxes for each object is predicted using a graph convolution network (GCN), which is then utilised to produce an image. Scene graphs, however, can be difficult to edit and do not allow for the direct specification of object locations on the image canvas.

3.4 Text to Image

Conditional image synthesis can be done in an intuitive way using textual descriptions. Current methods first produce a text embedding, which is then input into an image generator with multiple stages. In order to learn the features of individual objects and control object locations, additional layout information is used by adding an object pathway. It has been explored to break down the work into predicting a semantic layout from text and then generating images that are conditioned on both the text and the semantic layout. Other works focus on text-guided image manipulation and separating content from style. However, textual descriptions are difficult to obtain by and natural language can be ambiguous.

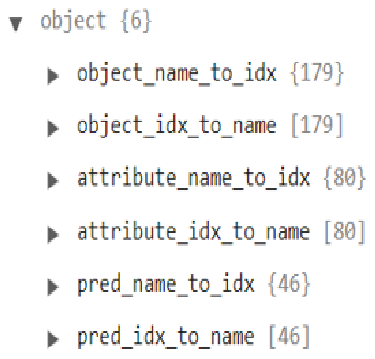
Chapter 4

Data Preprocessing

We need to preprocess the data. To clean the data, we will divide it into train, val, and test halves, consolidate all scene graphs into HDF5 files, using several heuristics. We specifically ignore images that are too small and only take into account object categories that appear a certain number of times in the training set. We also ignore objects that are too small and set minimum and maximum values for the number of objects and relationships that appear per image.

A vocab file is created containing visual genome vocabulary

1. Images which are too small in size are removed
2. vocab file for objects and relationships is generated by encoding objects and relationships



```
▼ object {6}
    ▶ object_name_to_idx {179}
    ▶ object_idx_to_name [179]
    ▶ attribute_name_to_idx {80}
    ▶ attribute_idx_to_name [80]
    ▶ pred_name_to_idx {46}
    ▶ pred_idx_to_name [46]
```

Figure 4.1: Description of how vocab.json created looks

Chapter 5

Layout to Image Synthesis

5.1 Introduction

We implement a GEnereative Adversarial Netwroks (GANs) and use a layout and style-based architecture and learning paradigm for GANs to learn controllable image synthesis from changeable layouts and style codes. This propsed method is termed as LostGAN.

By training GANs for layout to mask to image synthesis, the proposed LostGAN solves the layout to image synthesis problem. Learning how to use layout to mask is a simple intermediate step that has advantages in two folds for accounting for the gap between bounding boxes in a layout and underlying object shapes: It helps in attaining objects with finer grained style control in image synthesized. Additionally, it helps in the separation of the learning of the object's geometry and learning of object's appearance. Since object appearance is disregarded, the layout to mask generation process is easier than the direct layout to image synthesis. It also makes sense to incorporate a layout to mask component in the interim, driven by the excellent recent development on conditional picture synthesis from semantic label maps. The learning of mask to image synthesis can benefit from the best practises in conditional image synthesis utilising semantic label maps if reasonably good object masks can be inferred for an input layout. The development of two-stage generators is a naive approach that might lead to less practical solutions. We Use is a single-stage learning model(i.e., using a single generator).

5.2 Generator

There are three inputs to the generator:

1. A spatial layout, L consisting of a number of object bounding boxes in an image lattice
2. For style control, a latent vector, z_{img} at the image level is used, and
3. a concatenation vector that joins the label embedding vector of the layout's object instances and a collection of object latent vectors, $zobji$.

The style control of individual object instances is done through the object latent vectors. The generator uses a new feature normalisation approach based on (2) as its direct input for object-level style control while using (2) as its direct input for overall style control (3)

5.2.1 ISLA-Norm

The layout to mask to image synthesis pipeline in our LostGANs is presented, and to implement it, the Instance-Sensitive and layout Aware Feature Normalization (ISLA-Norm) approach is proposed. It consists of two parts as a feature normalisation scheme: feature standardisation and feature recalibration. The former is carried out using the BatchNorm, in which a mini-batch is used to compute the channel-wise mean and standard deviation. T he latter is different compared to the BatchNorm.

To improve style control and diversity, object latent vectors are used at every stage of the generation process, much as StyleGANs.

The ISLA Norm first learns object instance-sensitive channel-wise affine transformations from the concatenation of object label embedding and object style latent vectors, in contrast to the BatchNorm, which learns channel-wise affine transformation parameters, *beta* (for re-shifting) and *gamma* (for rescaling), as model parameters and shared across spatial dimensions by all instances. This is similar to the projection-based conditional BatchNorm in cGANs and the adaptive instance normalisation (AdaIN) used in StyleGANs. Better object masks are learnt by ISLA Norm for objects in an input layout using two different learning pathways. One pathway learns a label map from each layer in the generator, and the other pathway learns a label map from the concatenation vector between the object label embedding vector and the object style latent vectors. At one point in the generator, the inferred label map is a learnable weighted sum of the two label maps. Then, in order to obtain fine-grained spatially distributed multi-object style control for an input layout, we embed the object instance-sensitive channel-wise affine transformations in the learned label map. This generates the instance-sensitive and layout aware affine transformations for feature recalibration in the generator.

5.3 Discriminator

There are two inputs to the discriminator: an input image, either real or fake, and the corresponding spatial layout. There are three components in discriminator:

1. a ResNet is used as backbone,
2. an image head classifier using the extracted features to compute the image realness score (the higher the score is, the more real an image is), and
3. an object head classifier computes the realness scores for the object instances. Although we do not apply the likelihood-based learning method in training, the realness score can also be seen as playing as the negative energy in energy-based models.

By using the bounding box in a given layout, the RoIAlign operator computes the feature representation for an object instance .

5.4 Loss Function

The loss function consists of both image and object adversarial hinge loss terms (balanced by a trade-off parameter, λ). The hinge loss aims to improve a synthesised image's realness score from a real image by a predetermined margin. The hinge loss performs better in the two player min-max game setup of GANs to enforce both the generator and the discriminator to be more aggressive in synthesising images of greater quality

Chapter 6

Experimentation

6.1 Motivation

6.1.1 Using ResNet as Generator and backbone of Discriminator

In general, we believe that when it comes to convolutional neural networks (CNNs), "the deeper the model, the better it is." This makes sense since, because they have a larger parameter space to explore, the models should be more capable (more adaptable to any space). The performance appears to deteriorate when the depth is increased, though. Prior to 2015, this was one of the network bottlenecks. ResNet provides a residual learning architecture that makes it easier to train networks that are significantly deeper while maintaining performance. It has won the 1st place on the ILSVRC 2015 for classification localization task. For many visual recognition tasks, the depth of representation is very important, thus we used this deep representation to generate class activation maps that indicate the discriminating image regions used by the CNN used to identify that category.

6.2 Preliminaries

6.2.1 Generator Architecture

The Genereator is composed of a linear full-connected (FC) layer, a no. of residual building blocks (ResBlocks), which depend on the target resolution of the synthesised image, and a "ToRGB" module for the output of the image. The architecture is illustrated in detail below.

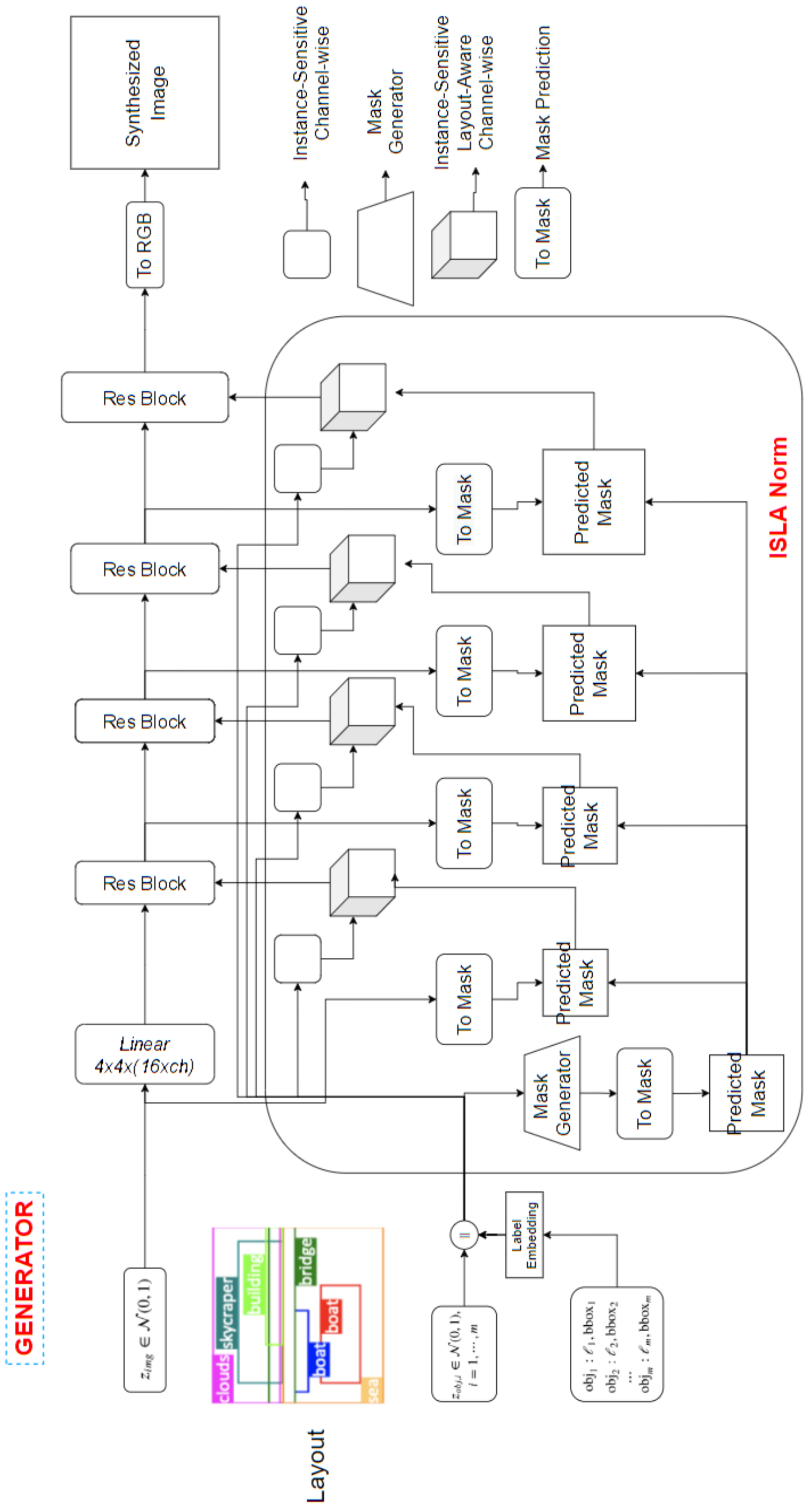


Figure 6.1: Generator Architecture

The ISLA Norm realises the learning of layout to mask to image synthesis. The intermediate masks help to improve the image synthesis output, leading to joint image and label map synthesis.

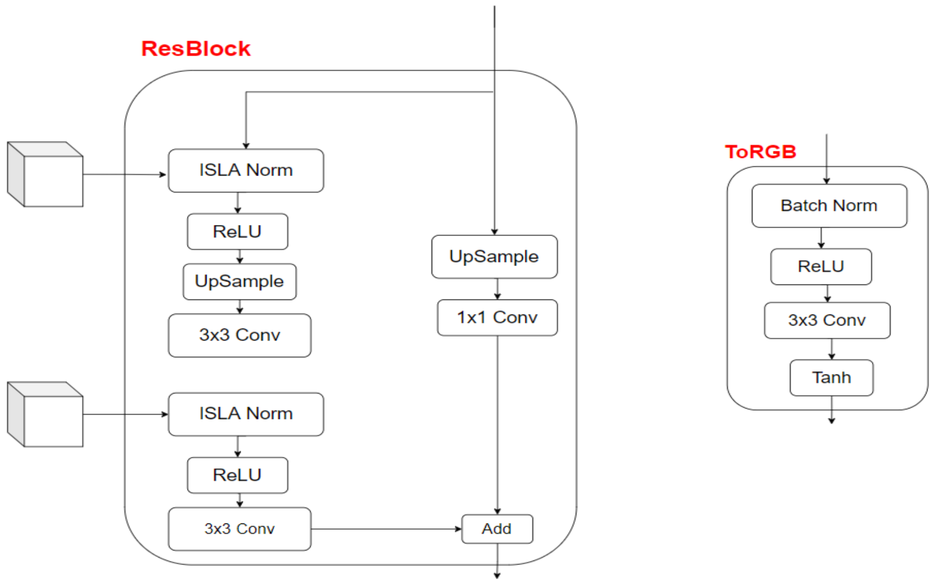


Figure 6.2: ResBlock in Generator Architecture

6.2.2 Discriminator Architecture

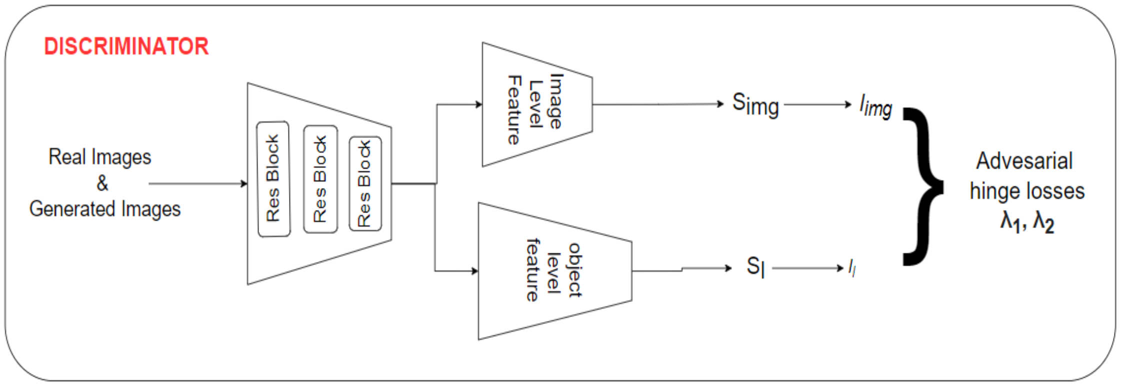


Figure 6.3: Discriminator Architecture

The image-level feature and the object-level feature share ResBlocks which depends on the target resolution

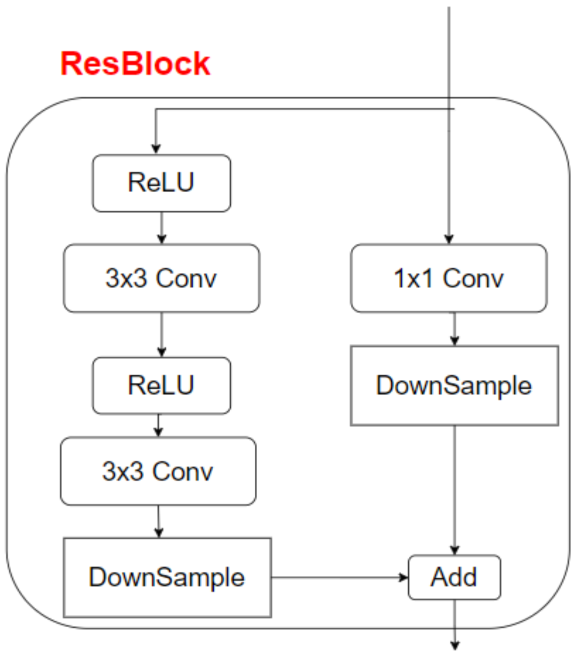


Figure 6.4: ResBlock in Discriminator Architecture

6.2.3 Methodolgy

There are two ISLA Norm modules in a ResBlock.ISLA Norm learns instance sensitive and layout aware affine transformation parameters, γ and β.
Computing γ & β is a multi step process:

1. *Label Embedding:* For each of the m object instances in the layout L, we apply one hot label vector, resulting in a one hot label matrix, represented by Y, of the size $m \times d_l$, where d_l is the number of object categories. Label embedding is the process of computing the vectorized representation for labels by learning an d_l x d_e embedding matrix, denoted by W.

$$\mathbf{Y} = Y.W$$

, $\mathbf{Y}$ is a m x d_e

2. *Joint Label and Style Encoding:* We sample from the object standard Gaussian distribution the object style latent codes Zobj which is a m×dobj noise matrix. Let S be the concatenation of label and style encoding,

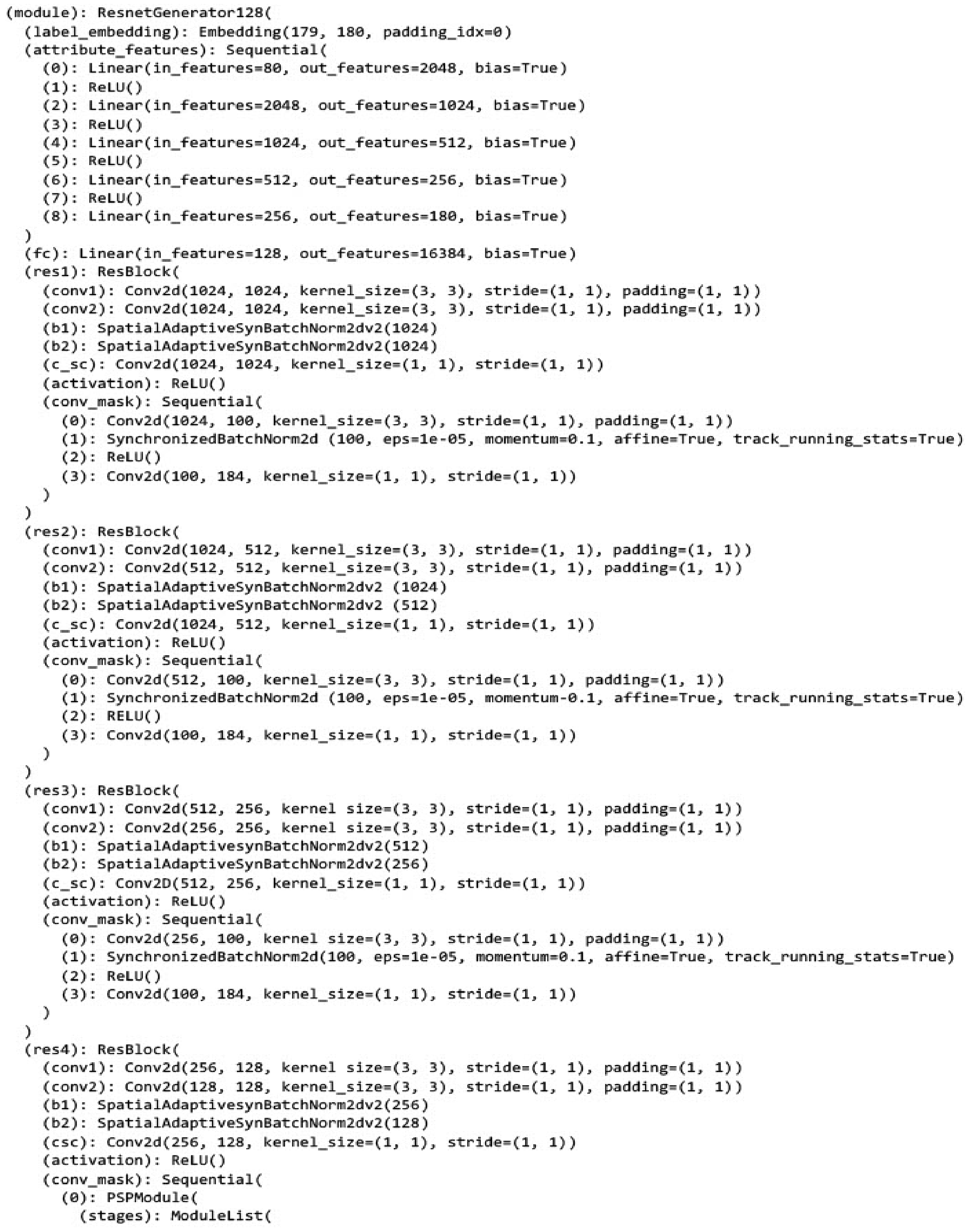
$$\mathbf{S} = (\mathbf{Y}, Z_{obj}),$$

which is a m x $(d_e + d_{obj}$ matrix.

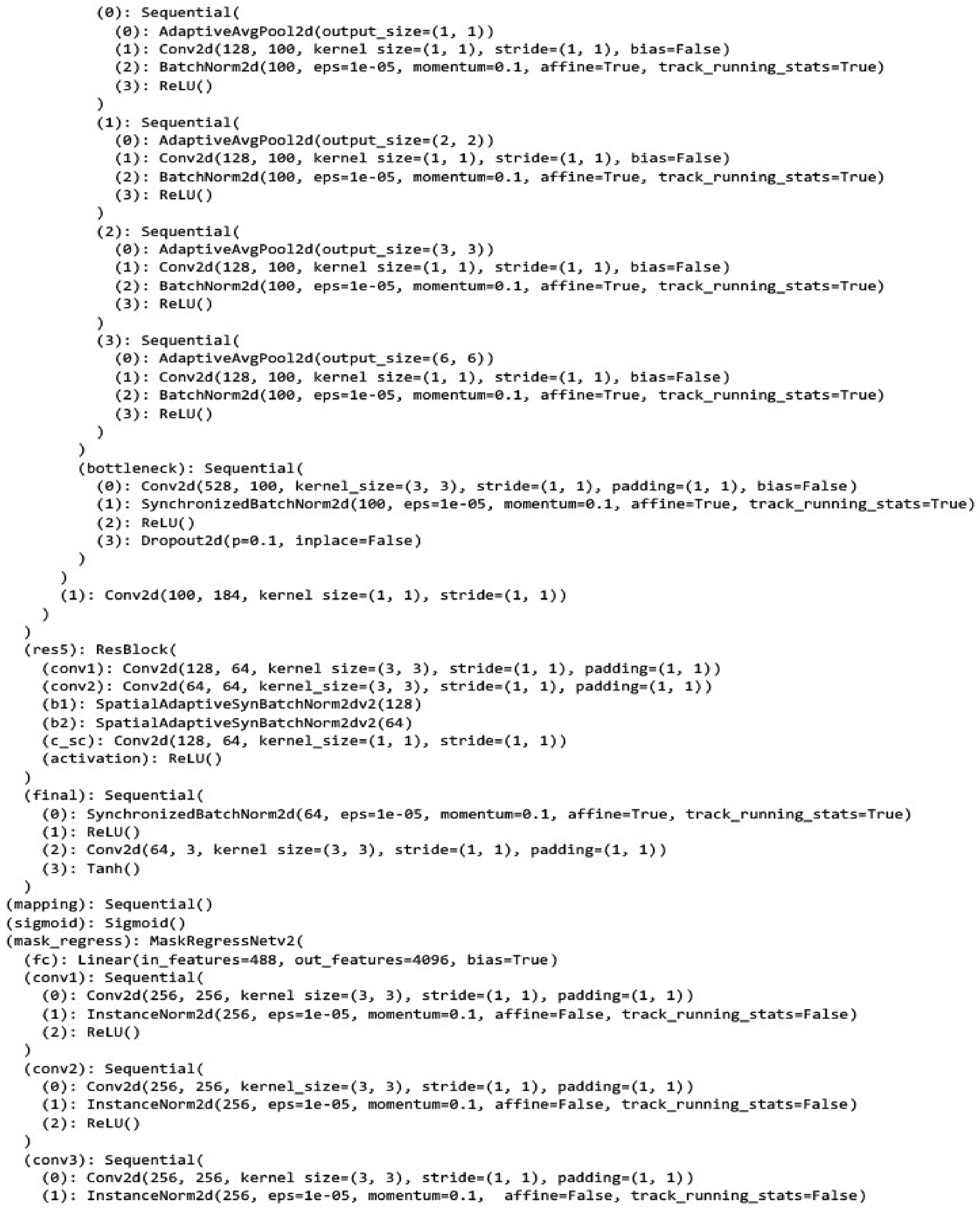
3. *Mask Prediction from S:* In the layout L, mask for each object is predicted first at a predefined size, p × p (e.g., p = 32) individually by a sub network made of up-sample convolutions and sigmoid transformation. Then the predicted masks are resized to the sizes of corresponding bounding boxes at a ResBlock stage in the generator.

4. *Computing ISLA γ and β:* The *gamma* and *beta* parameters are unsqueezed to their appropriate bounding boxes, weighted by the predicted masks, and lastly added with an averaged sum used for overlapping regions. LostGAN achieves better and more precise control over the image generation process since the affine transformation parameters depend on specific objects in a sample (class labels, bounding boxes, and styles). By changing the layout and sampling latent codes, the user is able to construct an image iteratively and interactively.

6.2.4 Experimental Setup:

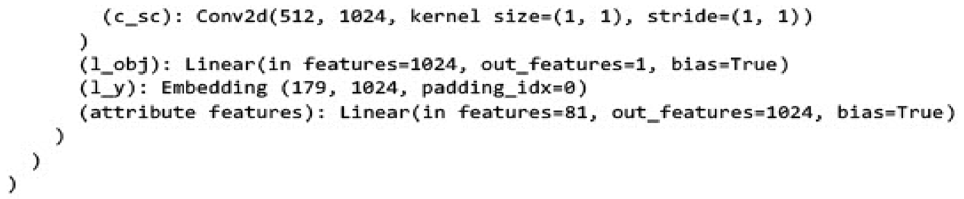


```
(module): ResnetGenerator128(
  (label_embedding): Embedding(179, 180, padding_idx=0)
  (attribute_features): Sequential(
    (0): Linear(in_features=80, out_features=2048, bias=True)
    (1): ReLU()
    (2): Linear(in_features=2048, out_features=1024, bias=True)
    (3): ReLU()
    (4): Linear(in_features=1024, out_features=512, bias=True)
    (5): ReLU()
    (6): Linear(in_features=512, out_features=256, bias=True)
    (7): ReLU()
    (8): Linear(in_features=256, out_features=180, bias=True)
  )
  (fc): Linear(in_features=128, out_features=16384, bias=True)
  (res1): ResBlock(
    (conv1): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (b1): SpatialAdaptiveSynBatchNorm2dv2(1024)
    (b2): SpatialAdaptiveSynBatchNorm2dv2(1024)
    (c_sc): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1))
    (activation): ReLU()
    (conv_mask): Sequential(
      (0): Conv2d(1024, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): SynchronizedBatchNorm2d (100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): Conv2d(100, 184, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (res2): ResBlock(
    (conv1): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (b1): SpatialAdaptiveSynBatchNorm2dv2 (1024)
    (b2): SpatialAdaptiveSynBatchNorm2dv2 (512)
    (c_sc): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1))
    (activation): ReLU()
    (conv_mask): Sequential(
      (0): Conv2d(512, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): SynchronizedBatchNorm2d (100, eps=1e-05, momentum-0.1, affine=True, track_running_stats=True)
      (2): RELU()
      (3): Conv2d(100, 184, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (res3): ResBlock(
    (conv1): Conv2d(512, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (b1): SpatialAdaptivesynBatchNorm2dv2(512)
    (b2): SpatialAdaptiveSynBatchNorm2dv2(256)
    (c_sc): Conv2D(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (activation): ReLU()
    (conv_mask): Sequential(
      (0): Conv2d(256, 100, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): SynchronizedBatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
      (3): Conv2d(100, 184, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (res4): ResBlock(
    (conv1): Conv2d(256, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (b1): SpatialAdaptivesynBatchNorm2dv2(256)
    (b2): SpatialAdaptiveSynBatchNorm2dv2(128)
    (csc): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
    (activation): ReLU()
    (conv_mask): Sequential(
      (0): PSPModule(
        (stages): ModuleList(
```



```
          (0): Sequential(
            (0): AdaptiveAvgPool2d(output_size=(1, 1))
            (1): Conv2d(128, 100, kernel size=(1, 1), stride=(1, 1), bias=False)
            (2): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): ReLU()
          )
          (1): Sequential(
            (0): AdaptiveAvgPool2d(output_size=(2, 2))
            (1): Conv2d(128, 100, kernel size=(1, 1), stride=(1, 1), bias=False)
            (2): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): ReLU()
          )
          (2): Sequential(
            (0): AdaptiveAvgPool2d(output_size=(3, 3))
            (1): Conv2d(128, 100, kernel size=(1, 1), stride=(1, 1), bias=False)
            (2): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): ReLU()
          )
          (3): Sequential(
            (0): AdaptiveAvgPool2d(output_size=(6, 6))
            (1): Conv2d(128, 100, kernel size=(1, 1), stride=(1, 1), bias=False)
            (2): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (3): ReLU()
          )
        )
        (bottleneck): Sequential(
          (0): Conv2d(528, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): SynchronizedBatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU()
          (3): Dropout2d(p=0.1, inplace=False)
        )
      )
      (1): Conv2d(100, 184, kernel size=(1, 1), stride=(1, 1))
    )
  )
  (res5): ResBlock(
    (conv1): Conv2d(128, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (b1): SpatialAdaptiveSynBatchNorm2dv2(128)
    (b2): SpatialAdaptiveSynBatchNorm2dv2(64)
    (c_sc): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
    (activation): ReLU()
  )
  (final): Sequential(
    (0): SynchronizedBatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): ReLU()
    (2): Conv2d(64, 3, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): Tanh()
  )
(mapping): Sequential()
(sigmoid): Sigmoid()
(mask_regress): MaskRegressNetv2(
  (fc): Linear(in_features=488, out_features=4096, bias=True)
  (conv1): Sequential(
    (0): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): ReLU()
  )
  (conv3): Sequential(
    (0): Conv2d(256, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1,  affine=False, track_running_stats=False)
    (2): ReLU()
    (3): Conv2d(256, 1, kernel_size=(1, 1), stride=(1, 1)),
    (4): Sigmoid()
  )
)


DataParallel (
  (module) : Combinediscriminator128(
    (obD): ResnetDiscriminator128(
      (block1): OptimizedBlock(
        (conv1): Conv2d(3, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1)),
        (conv2): Conv2d(64, 64, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (c_sc): Conv2d(3, 64, kernel size=(1, 1), stride=(1, 1))
        (activation): ReLU()
      )
      (block2): ResBlock(
        (conv1): Conv2d(64, 128, kernel size=(3, 3), stride=(1, 1), padding=(1, 1)),
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
        (activation): ReLU()
        (c_sc): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1))
      )
      (block3): ResBlock(
        (conv1): Conv2d(128, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): ReLU()
        (c_sc): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (block4): ResBlock(
        (conv1): Conv2d(256, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(512, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): ReLU()
        (c_sc): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
      )
      (block5): ResBlock(
        (conv1): Conv2d(512, 1024, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): ReLU()
        (c_sc): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
      )
      (block6): ResBlock(
        (conv1): Conv2d(1024, 1024, kernel size=(3, 3),
        stride=(1, 1), padding=(1, 1)),
        (conv2): Conv2d(1024, 1024, kernel size=(3, 3), stride=(1, 1), padding=(1, 1)),
        (activation): ReLU()
      )
      (l7): Linear(in features=1024, out_features=1, bias=True)
      (activation): ReLU()
      (roi_align_s): RolAlign(output_size=(8, 8), spatial_scale=0.25, sampling_ratio=0, aligned=False)
      (roi_align_l): RolAlign(output_size=(8, 8), spatial_scale=0.125, sampling_ratio=0, aligned=False)
      (block_obj3): ResBlock(
        (conv1): Conv2d(128, 256, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): ReLU()
        (c_sc): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (block_obj4): ResBlock(
        (conv1): Conv2d(256, 512, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): ReLU()
        (c_sc): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
      )
      (block_obj5): ResBlock(
        (conv1): Conv2d(512, 1024, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (Conv2): Conv2d(1024, 1024, kernel size=(3, 3), stride=(1, 1), padding=(1, 1))
        (activation): ReLU()
```



```
        (c_sc): Conv2d(512, 1024, kernel size=(1, 1), stride=(1, 1))
      )
      (l_obj): Linear(in features=1024, out_features=1, bias=True)
      (l_y): Embedding (179, 1024, padding_idx=0)
      (attribute features): Linear(in features=81, out_features=1024, bias=True)
    )
  )
)
```

6.2.5 Loss Function:

Below are the plots of changes in loss values at every step of epoch visualised on tensorboard :

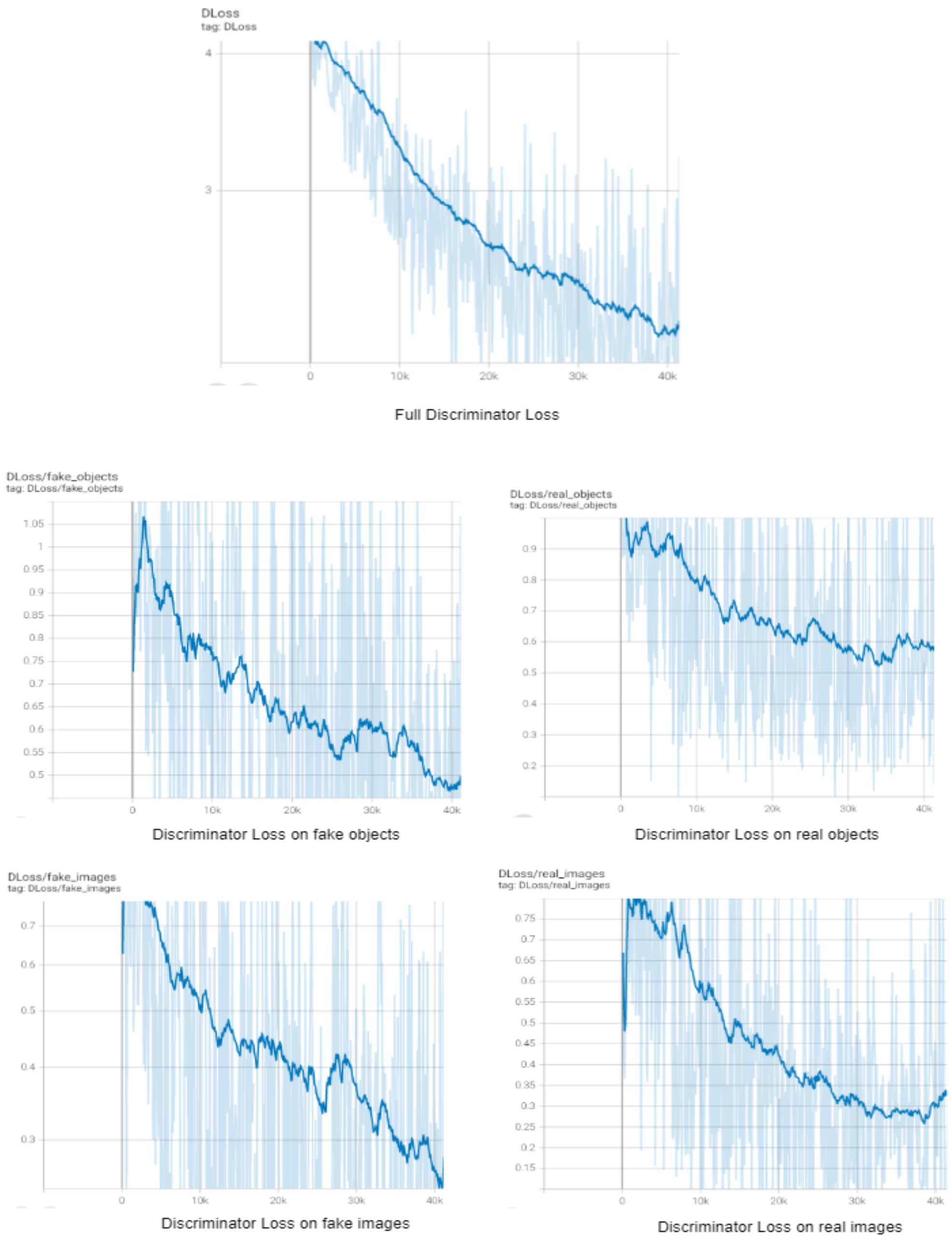


Figure 6.5: Discriminator Losses

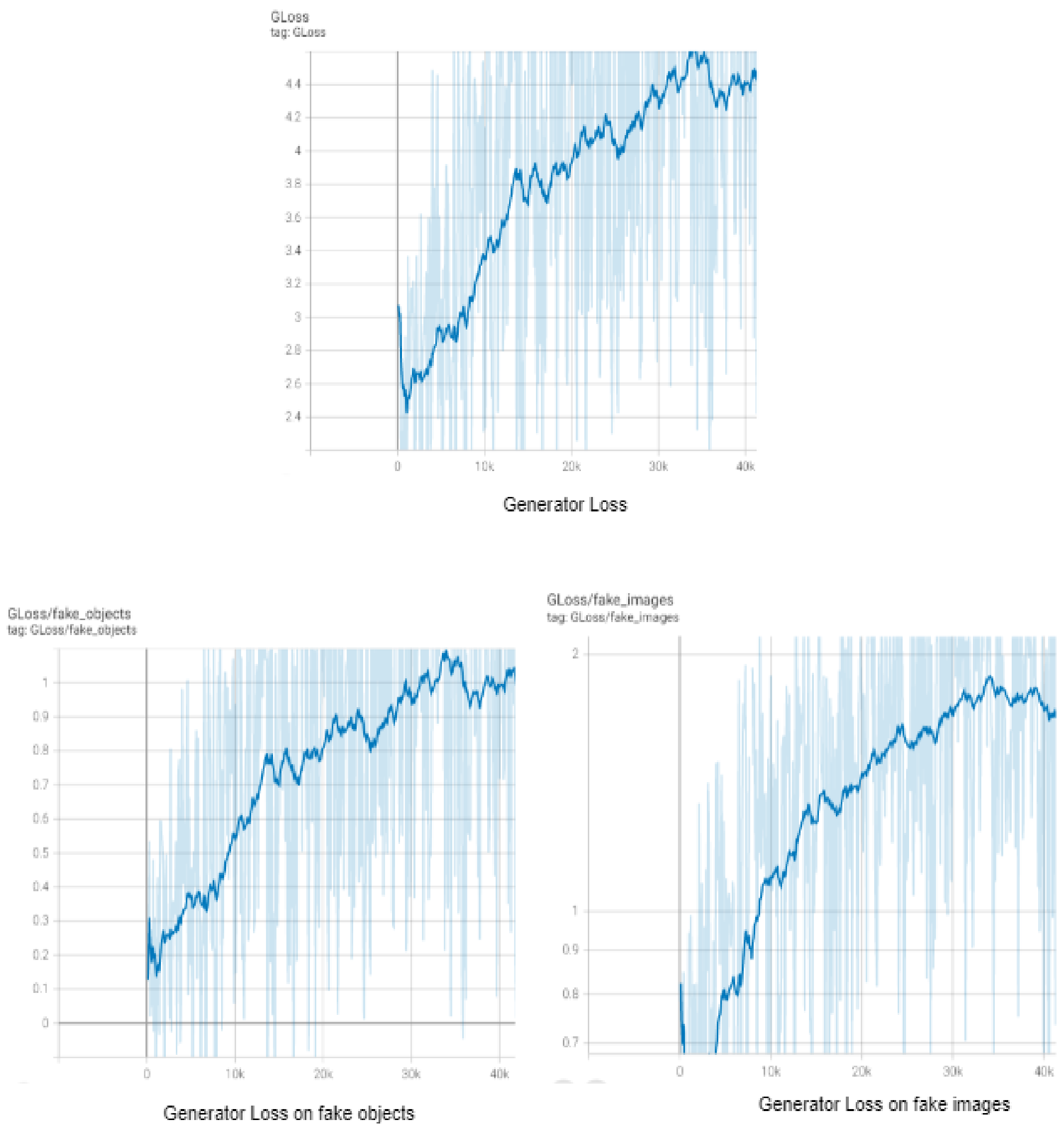
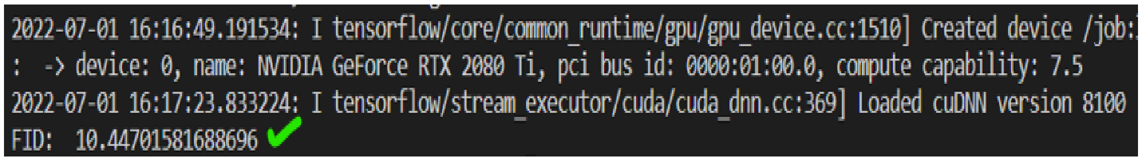


Figure 6.6: Generator Losses

6.2.6 Result

To compare the quality of the images synthesized on the built GAN, we use FID score (Frechet Inception Distance). we evaluate the FID between real images and 5061 fake generated images, the FID score came around 10.44.



```
2022-07-01 16:16:49.191534: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:
:  -> device: 0, name: NVIDIA GeForce RTX 2080 Ti, pci bus id: 0000:01:00.0, compute capability: 7.5
2022-07-01 16:17:23.833224: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8100
FID:  10.44701581688696 ✓
```

Figure 6.7: FID score

Below are comparison between samples of real images and fake images generated with image size set to 128 x 128:

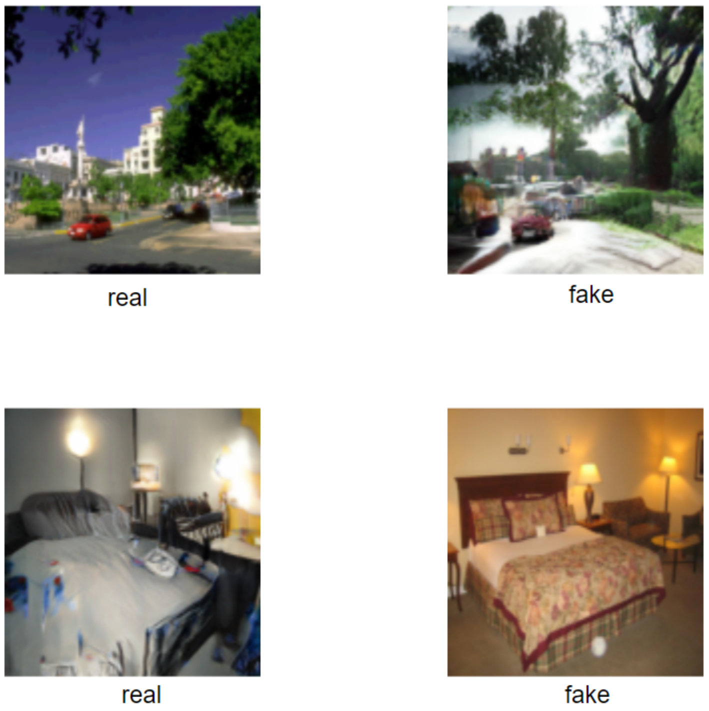


Figure 6.8

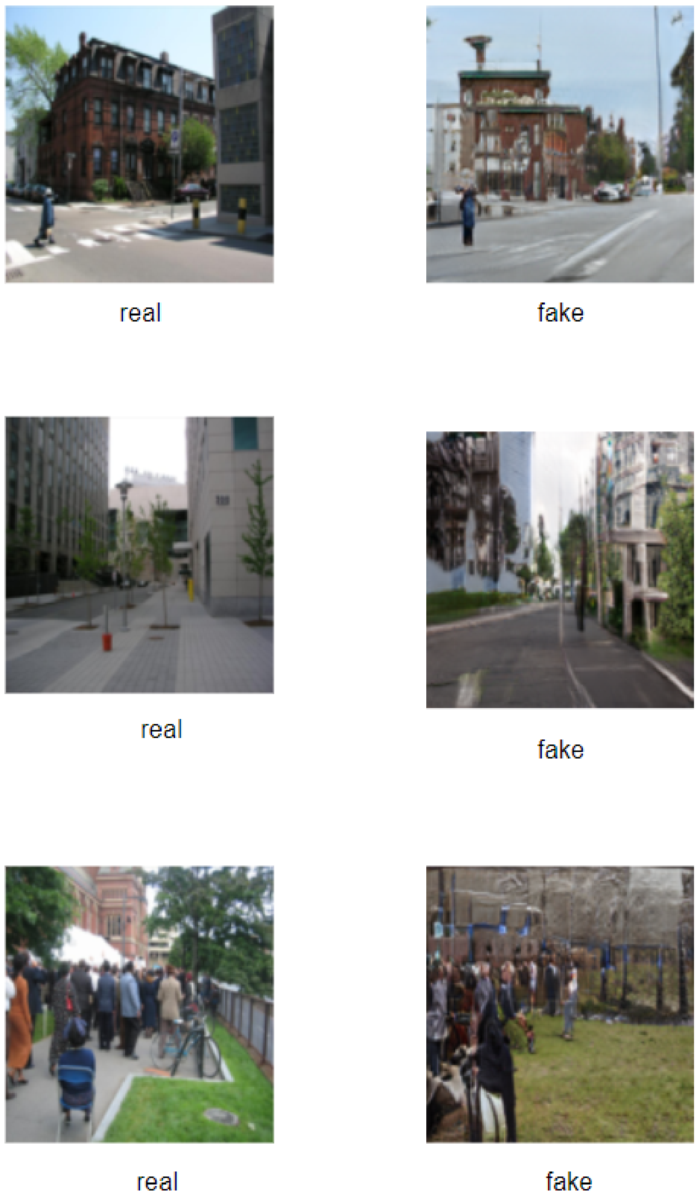


Figure 6.9

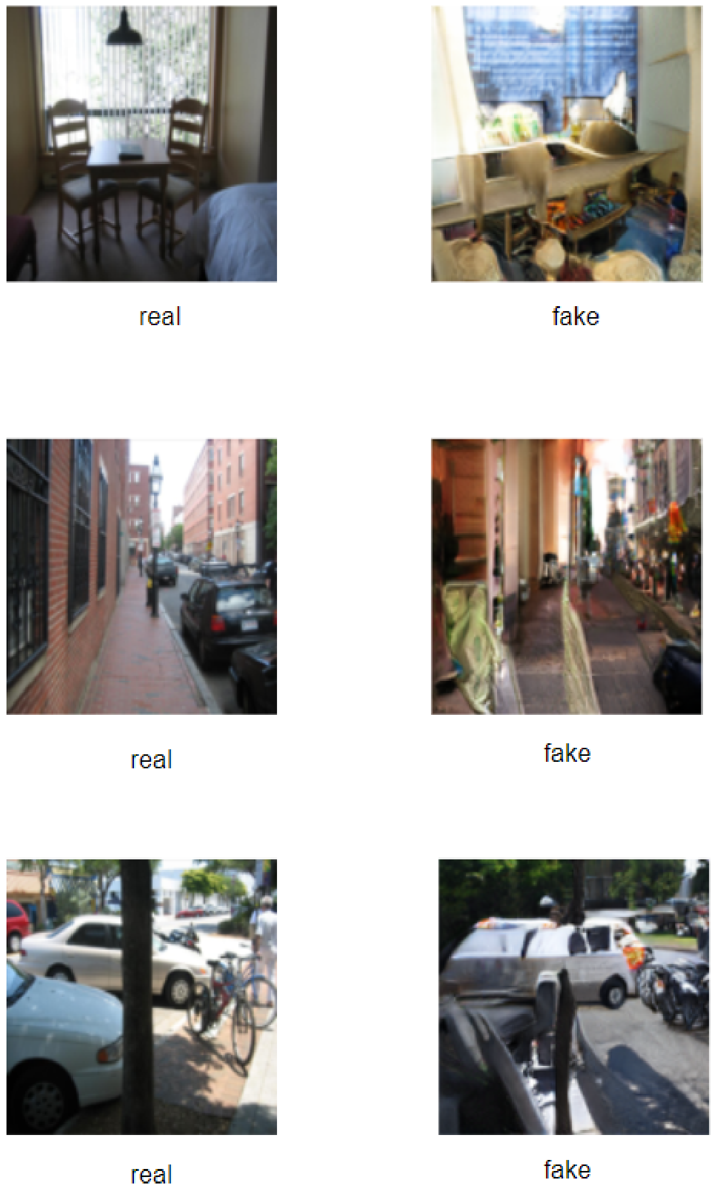


Figure 6.10: Comparison of real images and fake generated images

Results for Layout to image synthesis is provided by manually providing bounding boxes and mapping them to objects. Here one such result is generated.

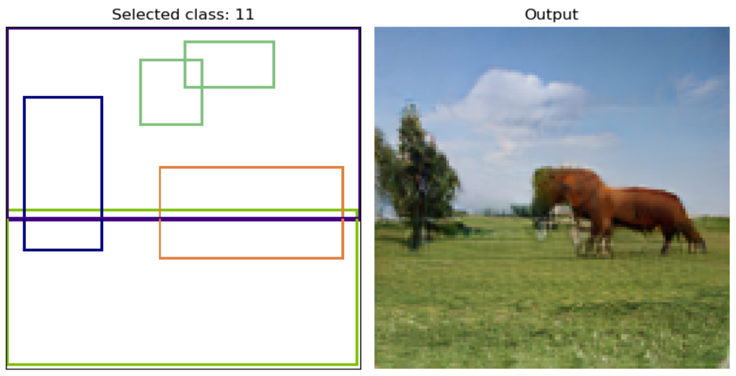


Figure 6.11: Layout to image generation

Chapter 7

Future Work and Conclusion

7.1 Conclusion

In this thesis, we study the generative learning problem of layout to image with a focus on controllable image synthesis from reconfigurable structured layouts and styles. We implement layout and style-based architecture for generative adversarial networks (termed LostGANs). On training the implemented GAN with image size as 128 x 128 and training for 200 epochs, we were able generate images with an FID score of 10.44 and further use the generator to perform image synthesis from layouts.

7.2 Future Work

The generative learning task of layout to image synthesis is still at a nascent stage of research in terms of synthesising high-fidelity images. Overall,on study it's found that the quality of image generation from layout is still not good enough.

Since each object in this method has a latent style code that is used to generate new images and is sampled from a normal distribution. There is lack of control which results in the inability to specify a certain style (for example, to change the colour of a jacket from red to yellow, one would need to sample new latent codes and manually inspect whether the generated style conforms to the requirement). To enable users to generate the images they imagine, it is essential to be able to control individual aspects of the generated image without affecting others.

For this to happen, we propose to introduce embedding for attributes specified for objects and concatenate them with label embedding and train the mask image synthesis component.

Bibliography

[1] Ashual, O., Wolf, L.: Specifying object attributes and relations in interactive scene generation (2019), https://arxiv.org/abs/1909.05379

[2] Dowson, D., Landau, B.: The frechet distance between multivariate normal distributions (1982)

[3] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks (2014), https://arxiv.org/abs/1406.2661

[4] Herzig, R., Bar, A., Xu, H., Chechik, G., Darrell, T., Globerson, A.: Learning canonical representations for scene graph to image generation (2019), https://arxiv.org/abs/1912.07414

[5] Johnson, J., Gupta, A., Fei-Fei, L.: Image generation from scene graphs (2018), https://arxiv.org/abs/1804.01622

[6] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.J., Shamma, D.A., Bernstein, M.S., Li, F.F.: Visual genome: Connecting language and vision using crowdsourced dense image annotations (2016), https://arxiv.org/abs/1602.07332

[7] Li, Z., Wu, J., Koh, I., Tang, Y., Sun, L.: Image synthesis from layout with locality-aware mask adaption. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 13799–13808 (2021)

[8] Ma, K., Zhao, B., Sigal, L.: Attribute-guided image generation from layout (2020), https://arxiv.org/abs/2008.11932

[9] Mirza, M., Osindero, S.: Conditional generative adversarial nets (2014), https://arxiv.org/abs/1411.1784

[10] Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational autoencoder for deep learning of images, labels and captions (2016), https://arxiv.org/abs/1609.08976

[11] Sun, W., Wu, T.: Image synthesis from reconfigurable layout and style (2019), `https://arxiv.org/abs/1908.07500`

[12] Zhao, B., Meng, L., Yin, W., Sigal, L.: Image generation from layout (2018), `https://arxiv.org/abs/1811.11389`