Towards Book Cover Design via Layout Graphs

Wensheng Zhang\textsuperscript{1}(✉), Yan Zheng\textsuperscript{1}, Taiga Miyazono\textsuperscript{1}, Seiichi Uchida\textsuperscript{1}[0000--0001--8592--7566], and Brian Kenji Iwana\textsuperscript{1}[0000--0002--5146--6818]

Kyushu University, Fukuoka, Japan
{zhang.wensheng,yan.zheng,taiga.miyazono}@human.ait.kyushu-u.ac.jp
{uchida,ivana}@ait.kyushu-u.ac.jp

Abstract. Book covers are intentionally designed and provide an introduction to a book. However, they typically require professional skills to design and produce the cover images. Thus, we propose a generative neural network that can produce book covers based on an easy-to-use layout graph. The layout graph contains objects such as text, natural scene objects, and solid color spaces. This layout graph is embedded using a graph convolutional neural network and then used with a mask proposal generator and a bounding-box generator and filled using an object proposal generator. Next, the objects are compiled into a single image and the entire network is trained using a combination of adversarial training, perceptual training, and reconstruction. Finally, a Style Retention Network (SRNet) is used to transfer the learned font style onto the desired text. Using the proposed method allows for easily controlled and unique book covers.

Keywords: Generative model · Book cover generation · Layout graph

1 Introduction

Book covers are designed to give potential readers clues about the contents of a book. As such, they are purposely designed to serve as a form of communication between the author and the reader [9]. Furthermore, there are many aspects of the design of a book cover that is important to the book. For example, the color of a book cover has shown to be a factor in selecting books by potential readers [13], the objects and photographs on a book cover are important for the storytelling [24], and even the typography conveys information [35,10]. Book covers [17,28] and the objects [21] on book covers also are indicators of genre.

While book cover design is important, book covers can also be time-consuming to create. Thus, there is a need for easy-to-use tools and automated processes which can generate book covers quickly. Typically, non-professional methods of designing book covers include software or web-based applications. There are many examples of this, such as Canva [6], fotor [11], Designhill [8], etc. These book cover designers either use preset templates or builders where the user selects from a set of fonts and images. The issue with these methods is that the design process is very restrictive and new designs are not actually created. It
is possible for multiple authors to use the same images and create similar book covers.

Recently, there has been an interest in machine learning-based generation. However, there are only a few examples of book cover-based generative models. In one example, the website deflamel \cite{7} generates designs based on automatically selected background and foreground images and title font. The images and font are determined based on a user-entered description of the book plus a “mood.” The use of Generative Adversarial Networks (GAN) \cite{12} have been used to generate books \cite{28,27}. Although, in the previous GAN-based generation methods, the created book covers were uncontrollable and generate gibberish text and scrambled images.

The problem with template-based methods is that new designs are not created and the problem with GAN-based methods is that it is difficult to control which objects are used and where they are located. Thus, we propose a method to generate book covers that addresses these problems. In this paper, we propose the use of a layout graph as the input for users to draw their desired book cover. The layout graph, as shown in Fig. 1, indicates the size, location, positional relationships, and appearance of desired text, objects, and solid color regions. The advantage of using the layout graph is that it is easy to describe a general layout for the proposed method to generate a book cover image from.

In order to generate the book cover image, the layout graph is provided to a generative network based on scene graph-based scene generators \cite{19,2}. In Fig. 2, the layout graph is fed to a Graph Convolution Network (GCN) \cite{32} to learn an embedding of the layout objects (i.e. text objects, scene objects, and solid regions). This embedding is used to create mask and bounding-box proposals using a mask generator and box regression network, respectively. Like \cite{2}, the mask proposals are used with an appearance generator to fill in the masks with contents. The generated objects are then aggregated into a single book cover image using a final generator. These generators are trained using four adversarial discriminators, a perception network, and L1 loss to a ground truth image. Finally, the learned text font is transferred to the desired text using a Style Retention Network (SRNet) \cite{37}.

The main contributions of this paper are summarized as follows:
As far as the authors know, this is the first instance of using a fully generative model for creating book cover images while being able to control the elements of the cover, such as size, location, and appearance of the text, objects, and solid regions.

We propose a method of using a combination of a layout graph-based generator and SRNet to create user-designed book cover images.

Our codes are shown at https://github.com/Touyuki/Cover_generation

2 Related Work

2.1 Document Generation

There are many generative models for documents. For example, automatic text and font generation is a key task in document generation. In the past, models such as using interpolation between multiple fonts [5,36] and using features from examples [34] have been used. More recently, the use of GANs have been used for font generation [1,14] and neural font style transfer [3] has become an especially popular topic in document generation.

There have also been attempts at creating synthetic documents using GANs [4,31] and document layout generation using recursive autoencoders [30]. Also, in a similar task to the proposed method, Hepburn et al. used a GAN to generate music album covers [16].

However, book cover generation, in particular, is a less explored area. Lucieri et al. [28] generated book covers using a GAN for data augmentation and the website Booksby.ai [27] generated entire books, including the cover, using GANs.
However, while the generated book covers have features of book covers and have the feel of book covers, the objects and text are completely unrecognizable and there is little control over the layout of the cover.

### 2.2 Scene Graph Generation

The proposed layout graph is based on scene graphs for natural scene generation. Scene graphs are a subset of knowledge graphs that specifically describe natural scenes, including objects and the relationships between objects. They were originally used for image retrieval [20] but were expanded to scene graph-based generation [19]. In scene graph generation, an image is generated based on the scene graph. Since the introduction of scene graph generation, there has been a huge boom of works in the field [38]. Some examples of scene graph generation with adversarial training, like the proposed method, include Scene Graph GAN (SG-GAN) [23], the scene generator by Ashual et al. [2], and PasteGAN [26]. These methods combine objects generated by each node of the scene graph and use a discriminator to train the scene image as a whole. As far as we know, we are the first to propose the use of scene graphs for documents.

### 3 Book Cover Generation

In this work, we generate book covers using a combination of two modules. The first is a Layout Generator. The Layout Generator takes a layout graph and translates it into an initial book cover image. Next, the neural font style transfer method, SRNet [37], is used to edit the generated placeholder text into a desired book cover text or title.

#### 3.1 Layout Generator

The purpose of the Layout Generator is to generate a book cover image including natural scene objects, solid regions (margins, headers, etc.), and the title text. To do this, we use a layout graph-based generator which is based on scene graph generation [19,2]. As shown in Fig. 2, the provided layout graph is given to a comprehensive model of an embedding network, four generators, four discriminators, and a perceptual consistency network. The output of the Layout Generator is a book cover image based on the layout graph.

**Layout Graph.** The input of the Layout Generator is a layout graph, which is a directed graph with each object \( o \) represented by a node \( \mathbf{n}_o = (\mathbf{c}_o, \mathbf{l}_o) \), where \( \mathbf{c}_o \) is a class vector and \( \mathbf{l}_o \) is the location vector of the object. The class vector contains a 128-dimensional embedding of the class of the object. The location vector \( \mathbf{l}_o \) is a 35-dimensional binary vector that includes the location and size of the object. The first 25 bits of \( \mathbf{l}_o \) describe the location of the object on a 5 \( \times \) 5 grid and the last 10 bits indicate the size of the desired object on a scale of 1 to 10.
Fig. 3. Illustration of the Graph Convolution Network. The red boxes are vertex vectors, the blue is the edge vector, the yellow is a hidden layer, and the arrows are full connections.

The edges of the layout graph are the positional relations between the objects. Each edge $e_{o,p}$ contains a 128-dimensional embedding of six relationships between every possible pairs of nodes $o$ and $p$. The six relationships include “right of”, “left of”, “above”, “below”, “surrounding” and “inside”.

**Graph Convolution Network.** The layout graph is fed to a GCN [32], $E_{\text{graph}}$, to learn an embedding $m_o$ of each object $o$.

Where a traditional Convolutional Neural Network (CNN) [25] uses convolutions of shared weights across an image, a GCN’s convolutional layers operate on graphs. They do this by traversing the graph and using a common operation on the edges of the graph.

To construct the GCN, we take the same approach as Johnson et al. [19] which constructs a list of all of the nodes and edges in combined vectors $v$ and then uses a multi-layer perceptron (MLP) on the vectors, as shown in Fig. 3. Vector $v$ consists of a concatenation of an edge embedding $e_{o,p}$ and the two adjacent vertices $o$ and $p$ and vertex embeddings $n_o$ and $n_p$. The GCN is consists of two sub-networks. The GCN (Edge) network in Fig. 3a takes in vector $v$ and then performs the MLP operation. The output is then broken up into temporary object segments $n'_o$ and $n'_p$ and further processed by individual GCN (Vertex) networks for each object. The result of GCN (Vertex) is a 128-dimensional embedding for each object, which is used by the subsequent Box Regression Network and Mask Generator.

**Mask Generator and Discriminator.** The purpose of the Mask Generator is to generate a mask of each isolated object for the Appearance Generator. The Mask Generator is based on a CNN. The input of the Mask Generator is the object embedding $m_o$ learned from the GCN and the output is a $32 \times 32$ shape mask of the target object. This mask is only the shape and does not include size
information. Furthermore, since the Mask Generator creates detailed masks, a variety of shapes should be used. To do this, a 64-dimensional random vector $z_o$ is concatenated with the object embedding $m_o$ before being given to the Mask Generator.

In order to produce realistic object masks, an adversarial Mask Discriminator $D_{mask}$ is used. The Mask Discriminator is based on a conditional Least Squares GAN (LS-GAN) [29] with the object class $s_o$ as the condition. It should be noted that the object class $s_o$ is different than the 128-dimensional class vector $c_o$ in the layout graph. The GAN loss $\mathcal{L}^D_{mask}$ is:

$$\mathcal{L}^D_{mask} = \log D_{mask}(T_o, s_o) + \mathbb{E}_{z_o \sim \mathcal{N}(0,1)^{64}} \log (1 - D_{mask}(G_{mask}(m_o, z_o), s_o)),$$

where $G_{mask}$ is the Mask Generator and $T_o$ is a real mask. Accordingly, the Mask Discriminator $D_{mask}$ is trained to minimize $-\mathcal{L}^D_{mask}$.

**Box Regression Network.** The Box Regression Network generates a bounding box estimation of where and how big the object should be placed in the layout. Just like the Mask Generator, the Box Regression Network receives the object embedding $m_o$. The Box Regression Network is an MLP that predicts the bounding box $b_o = \{(x_0, y_0), (x_1, y_1)\}$ coordinates for each object $o$.

To generate the layout, the outputs of the Mask Generator and the Box Regression Network are combined. In order to accomplish this, the object masks from the Mask Generator are shifted and scaled according to bounding boxes. The shifted and scaled object masks are then concatenated in the channel dimension and used with the Appearance Generator to create a layout feature map $F$ for the Book Cover Generator.

**Appearance Generator.** The objects’ appearances that are bound by the mask are provided by the Appearance Generator $G_{app}$. The Appearance Generator is a CNN that takes real images of cropped objects of (64 $\times$ 64 $\times$ 3) resolution and encodes the appearance into a 32-dimension appearance vector. The appearance vectors $a_o$ represent objects within the same class and changing the appearance vectors allows the appearance of the objects in the final generated result to be controlled. This gives the network to provide a variety of different object appearances even with the same layout graph. A feature map $F$ is created by compiling the appearance vectors to fill the masks that were shifted and scaled by the bounding boxes.

**Book Cover Generator.** The Book Cover Generator $G_{book}$ is based on a deep Residual Network (ResNet) [15] and it generates the final output. The network has three parts. The first part is the contracting path made of strided convolutions which encodes the features from the feature map $F$. The second part is a series of 10 residual blocks and the final part is an expanding path with transposed convolutions to upsample the features to the final output image $I$. 
**Perception Network.** In order to enhance the quality of output of the Book Cover Generator a Perception Network is used. The Perception Network $P_{\text{content}}$ is a pre-trained very deep convolutional network (VGG) [33] that is only used to establish a perceptual loss $L^P_{\text{content}}$ [18]. The perceptual loss:

$$
L^P_{\text{content}} = \sum_{u \in U} \frac{1}{u} \left| P^{(u)}_{\text{content}}(I) - P^{(u)}_{\text{content}}(R) \right|
$$

is the content consistency between the extracted features of the VGG network $P_{\text{content}}$ given the generated layout image $I$ and a real layout image $R$. In Eq. (2), $u$ is a layer in the set of layers $U$ and $P^{(u)}_{\text{content}}$ is a feature map from $P_{\text{content}}$ at layer $u$.

**Layout Discriminator.** The Layout Discriminator $D_{\text{layout}}$ is a CNN used to judge whether the layout image $I$ appears realistic given the layout $F$. In this way, through the compound adversarial loss $L_{\text{layout}}$, the generated layout will be trained to be more indistinguishable from images of real layout images $R$ and real layout feature maps $Q$. The loss $L_{\text{layout}}$ is defined as:

$$
L_{\text{layout}} = \log D_{\text{layout}}(Q, R) + \log(1 - D_{\text{layout}}(Q, I)) + \log(1 - D_{\text{layout}}(F, R)) + \log D_{\text{layout}}(Q', R)
$$

where $Q'$ is a second layout with the bounding box, mask, and appearance attributes taken from a different, incorrect ground truth image with the same objects. This is used as a poor match despite having the correct objects. The aim of the Layout Discriminator is to help the generated image $I$ with ground truth layout $Q$ to be indistinguishable from real image $R$.

**Book Cover Discriminator.** The Book Cover Discriminator is an additional discriminator that is used to make the generated image look more like a book. Unlike the Layout Discriminator, the Book Cover Discriminator only compares the generated image $I$ to random real book covers $B$. Specifically, an adversarial loss:

$$
L_{\text{book}} = \log D_{\text{book}}(B) + \log(1 - D_{\text{book}}(I)),
$$

where $D_{\text{book}}$ is the Book Cover Discriminator, is added to the overall loss.

**Object Discriminator.** The Object Discriminator $D_{\text{obj}}$ is another CNN used to make each object images look real. $i_o$ is an object image cut from the generated image by the generated bounding box and $r_o$ is a real crop from the ground truth image. The object loss $L_{\text{obj}}$ is:

$$
L_{\text{obj}} = \sum_{o=1}^{O} \log D_{\text{obj}}(r_o) - \log D_{\text{obj}}(i_o).
$$
Training. The entire Layout Generator with all the aforementioned networks are trained together end-to-end. This is done using a total loss:

\[
\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{pixel}} + \lambda_2 \mathcal{L}_{\text{box}} + \lambda_3 \mathcal{L}_{\text{content}} + \lambda_4 \mathcal{L}_{\text{D mask}} + \lambda_5 \mathcal{L}_{\text{D obj}} + \lambda_6 \mathcal{L}_{\text{D layout}} + \lambda_7 \mathcal{L}_{\text{D book}} + \lambda_8 \mathcal{L}_{\text{P mask}} + \lambda_9 \mathcal{L}_{\text{P layout}},
\]

(6)

where each \( \lambda \) is a weighting factor for each loss. In addition to the previously described losses, Eq. (6) contains a pixel loss \( \mathcal{L}_{\text{pixel}} \) and two additional perceptual losses \( \mathcal{L}_{\text{P mask}} \) and \( \mathcal{L}_{\text{P layout}} \). The pixel loss \( \mathcal{L}_{\text{pixel}} \) is the L1 distance between the generated image \( I \) and the ground truth image \( R \). The two perceptual losses \( \mathcal{L}_{\text{P mask}} \) and \( \mathcal{L}_{\text{P layout}} \) are similar to \( \mathcal{L}_{\text{P content}} \) (Eq. 2), except instead of a separate network, the feature maps of all of the layers of discriminators \( D_{\text{mask}} \) and \( D_{\text{layout}} \) are used, respectively.

3.2 Solid Region Generation

The original scene object generation is designed to generate objects in natural scenes that seem realistic. However, if we want to use it in book cover generation we should generate more elements that are unique to book covers, such as the solid region and the title information.

We refer to solid regions as regions on a book with simple colors. They can be a single solid color, gradients, or subtle designs. As shown in Fig. 8, they are often used for visual saliency, backgrounds, and text regions. Except for some text information, usually, there are no other elements in these regions. To incorporate the solid regions into the proposed model, we prepared solid regions as objects in the Layout Graph. In addition, the solid regions are added as an object class to the various components of the Layout Generator as well as added to the ground truth images \( R \) and layout feature maps \( Q \). To make sure we can generate realistic solid regions, in our experiment, we used solid regions cut from real book covers.

3.3 Title Text Generation

Text information is also an important part of the book covers. It contains titles, sub-titles, author information, and other text. In our experiment, we only consider the title text.
Unlike other objects, like trees, the text information cannot be random variations and has to be determined by the user. However, the text still needs to maintain a style and font that is suitable for the book cover image.

Thus, we propose to generate the title text in the image using a placeholder and use font style transfer to transfer the placeholder’s font to the desired text. Fig. 5 shows our process of transferring the font style to the title text. To do this, we use SRNet [37]. SRNet is a neural style transfer method that uses a skeleton-guided network to transfer the style of text from one image to another. In SRNet, there are two inputs, the desired text in a plain font and the stylized text. The two texts are fed into a multi-task encoder-decoder that generates a skeleton image and a stylized image of the desired text. Using SRNet, we can generate any text using the style learned by the Layout Generator and use it to replace the placeholder.

To train the Layout Generator, we use a placeholder text, “Lorem Ipsum,” to represent the title. Similar to the solid region object, the title object is also added as an object class. For the ground truth images, a random font, color, and location are used. However, the purpose of the Book Cover Discriminator $D_{book}$ is to ensure that the combination is realistic as books.

4 Experimental Results

4.1 Dataset

To train the proposed method two datasets are required. The first is the Book Cover Dataset\(^1\). This dataset is made of book cover images and is used to train the Book Cover Discriminator.

For the second dataset, a natural scene object dataset with semantic segmentation information is required. For this, we use 5,000 images from the COCO\(^2\) dataset. For the ground truth images and layouts, random solid regions and titles are added. The cropped parts of COCO are used with the Mask Discriminator

---

\(^1\) [https://github.com/uchidalab/book-dataset](https://github.com/uchidalab/book-dataset)

\(^2\) [https://cocodataset.org/](https://cocodataset.org/)
and the Object Discriminator, and modified images of COCO are used for the Layout Discriminator and Perception Network. All of the images are resized to $128 \times 128$.

### 4.2 Settings and Architecture

The networks in the Layout Generator are trained end-to-end using Adam optimizer [22] with $\beta = 0.5$ and an initial learning rate of 0.001 for 100,000 iterations with batch size 6. For the losses, we set $\lambda_1 = 1$, $\lambda_4 = 1$, $\lambda_6 = 1$, $\lambda_7 = 1$, $\lambda_2 = 10$, $\lambda_3 = 10$, $\lambda_8 = 10$, and $\lambda_9 = 0.1$. The hyperparameters used in the experiments are listed in Table 1. For SRNet, we used a pre-trained model\(^3\).

\(^3\) [https://github.com/Niwhskal/SRNet](https://github.com/Niwhskal/SRNet)
Table 1. The architecture of the networks.

| Network                          | Layers                                      | Activation | Norm.      |
|----------------------------------|---------------------------------------------|------------|------------|
| GCN (Edge)                      | FC, 512 nodes                               | ReLU       |            |
|                                  | FC, 1,152 nodes                             | ReLU       |            |
| GCN (Vertex)                    | FC, 512 nodes                               | ReLU       |            |
|                                  | FC, 128 nodes                               | ReLU       |            |
| Box Regression Network          | FC, 512 nodes                               | ReLU       |            |
|                                  | FC, 4 nodes                                 | ReLU       |            |
| Mask Generator                  | Conv. (3 x 3), 192 filters, stride 1        | ReLU       | Batch norm |
|                                  | Conv. (3 x 3), 192 filters, stride 1        | ReLU       | Batch norm |
|                                  | Conv. (3 x 3), 192 filters, stride 1        | ReLU       | Batch norm |
|                                  | Conv. (3 x 3), 192 filters, stride 1        | ReLU       | Batch norm |
| Appearance Generator            | Conv. (4 x 4), 64 filters, stride 2         | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 128 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 256 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Global Average Pooling                      | ReLU       |            |
|                                  | FC, 192 nodes                               | ReLU       |            |
|                                  | FC, 64 nodes                                | ReLU       |            |
| Book Cover Generator            | Conv. (7 x 7), 64, stride 1                 | ReLU       | Inst. norm |
|                                  | Conv. (3 x 3), 128 filters, stride 2        | ReLU       | Inst. norm |
|                                  | Conv. (3 x 3), 256 filters, stride 2        | ReLU       | Inst. norm |
|                                  | Conv. (3 x 3), 512 filters, stride 2        | ReLU       | Inst. norm |
|                                  | Conv. (3 x 3), 1,024 filters, stride 2      | ReLU       | Inst. norm |
|                                  | Conv. (3 x 3), 1,024 filters, stride 1      | ReLU       | Inst. norm |
|                                  | T. conv. (3 x 4), 512 filters, stride 2     | ReLU       | Inst. norm |
|                                  | T. conv. (3 x 3), 256 filters, stride 2     | ReLU       | Inst. norm |
|                                  | T. conv. (3 x 3), 128 filters, stride 2     | ReLU       | Inst. norm |
|                                  | T. conv. (3 x 3), 64 filters, stride 2      | ReLU       | Inst. norm |
| (x10 residual blocks)           | Conv. (7 x 7), 3 filters, stride 1          | ReLU       |           |
| Mask Discriminator              | Conv. (3 x 3), 64 filters, stride 2         | LeakyReLU  | Inst. norm |
|                                  | Conv. (3 x 3), 128 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (3 x 3), 256 filters, stride 1        | LeakyReLU  | Inst. norm |
|                                  | Conv. (3 x 3), 1 filters, stride 1          | LeakyReLU  | Inst. norm |
|                                  | Ave. Pooling (3 x 3), stride 2              | LeakyReLU  |           |
| Layout Discriminator            | Conv. (4 x 4), 64 filters, stride 2         | LeakyReLU  |           |
|                                  | Conv. (4 x 4), 128 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (4 x 4), 256 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (4 x 4), 512 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (4 x 4), 1 filter, stride 2           | Linear     |            |
|                                  | Conv. (4 x 4), 64 filters, stride 2         | LeakyReLU  |           |
|                                  | Conv. (4 x 4), 128 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (4 x 4), 256 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (4 x 4), 512 filters, stride 2        | LeakyReLU  | Inst. norm |
|                                  | Conv. (4 x 4), 1 filter, stride 2           | Linear     |            |
|                                  | Ave. Pooling (3 x 3), stride 2              | LeakyReLU  |           |
| Book Cover Discriminator        | Conv. (4 x 4), 64 filters, stride 2         | LeakyReLU  |           |
|                                  | Conv. (4 x 4), 128 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 256 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 512 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 1 filter, stride 2           | Sigmoid    |            |
| Object Discriminator            | Conv. (4 x 4), 64 filters, stride 2         | LeakyReLU  |           |
|                                  | Conv. (4 x 4), 128 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 256 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 512 filters, stride 2        | LeakyReLU  | Batch norm |
|                                  | Conv. (4 x 4), 1 filter, stride 2           | Sigmoid    |            |
| Perception Network              | Pre-trained VGG [33]                        |            |            |
| Font Style Transfer             | Pre-trained SRNet [37]                      |            |            |
4.3 Generation Results

Examples of generated book covers are shown in Fig. 6. We can notice that not only the object images can be recognizable, but also the solid regions make the results resemble book covers. In addition, for most of the results, the generated titles are legible. While not perfect, these book covers are a big step towards book cover generation. We also shows some images with poor quality in Fig. 7. In these results the layout maps are reasonable, but the output is still poor. This is generally due to having overlapping objects such as “grass” on the “title” or objects overlapping the solid regions.

4.4 Creating Variations in Book Covers

As mentioned previously, the advantage of using a layout graph is that each node contains information about the object, location, and appearance embedding. This allows for the ease of book cover customization using an easy to use interface. Thus, we will discuss some of the effects of using the layout graph to make different book cover images.

**Location on the solid region.** Along with the scene objects, the title text and the solid region can be moved on the layout graph. Fig. 8 shows examples of generated book covers with the same layout graph except for the “Solid” nodes. By moving the “Solid” node to have different relationship edges with other nodes, the solid regions can be moved. In addition, multiple “Solid” nodes can be added to the same layout graph to construct multiple solid regions.

**Variation in the appearance vector.** Due to each node in the layout graph containing its own appearance vector, different variations of generated book covers can be created from the same layout graph. Fig. 9 shows a layout graph and the effects of changing the appearance vector of individual nodes. In the figure, only one node is changed and all the rest are kept constant. However, even though only one element is being changed in each sub-figure, multiple elements are affected. For example, in Fig. 9 (c) when changing the “Grass” node, the generated grass area changes and the model automatically changes the “Solid” and “Sky” regions to match the appearance of the “Grass” region. As it can

![Fig. 7. Example of poor results.](image-url)
be observed from the figure, the solid bar on the left is normally contrasted from the sky and the grass. This happens because each node is not trained in isolation and the discriminators have a global effect on multiple elements and aim to generate more realistic compositions.

4.5 Effect of Text Style Transfer

The SRNet is used to change the placeholder text to the desired text in the generated image. In Fig. 10, we show a comparison of book covers before and after using SRNet. As can be seen from the figure, SRNet is able to successfully transfer the font generated by the Layout Generator and apply it to the desired text. This includes transferring the color and font features of the placeholder. In addition, even if the title text is short like “Sheep” or “Color,” SRNet was able to still erase the longer placeholder text. However, “Winter Day” appears to erroneously overlap with the solid region, but that is due to the predicted bounding box of the text overlapping with the solid region. Thus, this is not a result of a problem with SRNet, but with the Box Regression Network.

5 Conclusion

We proposed a book cover image generation system given a layout graph as the input. It comprises an image generation model and a font style transfer network. The image generation model uses a combination of a GCN, four generators, four discriminators, and a perception network to a layout image. The font style transfer network then transfers the style of the learned font onto a replacement with the desired text. This system allows the user to control the book cover elements and their sizes, locations, and appearances easily. In addition, users can write any text information and fonts fitting the book cover will be generated. Our research is a step closer to automatic book cover generation.
Fig. 9. Examples of the effect of changing the appearance vector for different nodes. Each sub-figure changes the appearance vector for the respective node and keeps all other nodes constant.

Acknowledgement

This work was in part supported by MEXT-Japan (Grant No. J17H06100 and Grant No. J21K17808).

References

1. Abe, K., Iwana, B.K., Holmer, V.G., Uchida, S.: Font creation using class discriminative deep convolutional generative adversarial networks. In: ACPR (2017)
2. Ashual, O., Wolf, L.: Specifying object attributes and relations in interactive scene generation. In: ICCV (2019)
3. Atarsaikhan, G., Iwana, B.K., Narusawa, A., Yanai, K., Uchida, S.: Neural font style transfer. In: ICDAR (2017)
4. Bui, Q.A., Mollard, D., Tabbone, S.: Automatic synthetic document image generation using generative adversarial networks: Application in mobile-captured document analysis. In: ICDAR (2019)
Towards Book Cover Design via Layout Graphs

Fig. 10. Using SRNet to change the placeholder title into a target text. The top row is the output before SRNet and the bottom is after SRNet.

5. Campbell, N.D.F., Kautz, J.: Learning a manifold of fonts. ACM Tran. Graphics 33(4), 1–11 (jul 2014)
6. Canva: Canva - the free book cover maker with stunning layouts. https://www.canva.com/create/book-covers/, accessed: 2021-02-15
7. Deflamel Corp.: deflamel. https://deflamel.com/, accessed: 2021-02-15
8. Designhill.com: Designhill - book cover maker. https://www.designhill.com/tools/book-cover-maker, accessed: 2021-02-15
9. Drew, N., Stemberge, P.: By Its Cover: Modern American Book Cover Design. Princeton Architectural Press (2005)
10. EL-Sakran, T., Ankit, A.: Representing academic disciplines on academic book covers. Int. J. Pedagogical Innov. 6(02), 151–163 (2018)
11. Everimaging Limited: fotor - book cover maker. https://www.fotor.com/design/book-cover, accessed: 2021-02-15
12. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NeurIPS (2014)
13. Gudinavičius, A., Šuminas, A.: Choosing a book by its cover: analysis of a reader’s choice. J. Doc. (2018)
14. Hayashi, H., Abe, K., Uchida, S.: GlyphGAN: Style-consistent font generation based on generative adversarial networks. Knowl.-Based Sys. 186, 104927 (dec 2019)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
16. Hepburn, A., McConville, R., Santos-Rodriguez, R.: Album cover generation from genre tags. In: MUSML (2017)
17. Iwana, B.K., Raza Rizvi, S.T., Ahmed, S., Dengel, A., Uchida, S.: Judging a book by its cover. arXiv preprint arXiv:1610.09204 (2016)
18. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: ECCV (2016)
19. Johnson, J., Gupta, A., Fei-Fei, L.: Image generation from scene graphs. In: CVPR (2018)
20. Johnson, J., Krishna, R., Stark, M., Li, L.J., Shamma, D.A., Bernstein, M.S., Fei-Fei, L.: Image retrieval using scene graphs. In: CVPR (2015)
21. Jolly, S., Iwana, B.K., Kuroki, R., Uchida, S.: How do convolutional neural networks learn design? In: ICPR (2018)
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Klawonn, M., Heim, E.: Generating triples with adversarial networks for scene graph construction. In: AAAI (2018)
24. Kratz, C.A.: On telling/selling a book by its cover. Cultur. Anthropol. 9(2), 179–200 (1994)
25. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE 86(11), 2278–2324 (1998)
26. Li, Y., Ma, T., Bai, Y., Duan, N., Wei, S., Wang, X.: Pastegan: A semi-parametric method to generate image from scene graph. In: NeurIPS (2019)
27. Loose, M.T., Refsgaard, A.: Booksby.ai - about. https://booksby.ai/about/, accessed: 2021-02-15
28. Lucieri, A., Sabir, H., Siddiqui, S.A., Rizvi, S.T.R., Iwana, B.K., Uchida, S., Dengel, A., Ahmed, S.: Benchmarking deep learning models for classification of book covers. SN Computer Sci. 1(3) (apr 2020)
29. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Paul Smolley, S.: Least squares generative adversarial networks. In: CVPR (2017)
30. Patil, A.G., Ben-Eliezer, O., Perel, O., Averbuch-Elor, H.: READ: Recursive autoencoders for document layout generation. In: CVPR Workshops (2020)
31. Rusticus, D., Goldmann, L., Reisser, M., Villegas, M.: Document domain adaptation with generative adversarial networks. In: ICDAR (2019)
32. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Networks 20(1), 61–80 (jan 2009)
33. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
34. Suveeranont, R., Igarashi, T.: Example-based automatic font generation. In: Smart Graphics (2010)
35. Tschichold, J.: The new typography: A handbook for modern designers, vol. 8. University of California Press (1998)
36. Uchida, S., Egashira, Y., Sato, K.: Exploring the world of fonts for discovering the most standard fonts and the missing fonts. In: ICDAR (2015)
37. Wu, L., Zhang, C., Liu, J., Han, J., Liu, J., Ding, E., Bai, X.: Editing text in the wild. In: ACM ICM (2019)
38. Xu, P., Chang, X., Guo, L., Huang, P.Y., Chen, X., Hauptmann, A.G.: A survey of scene graph: Generation and application. EasyChair Preprint (2020)