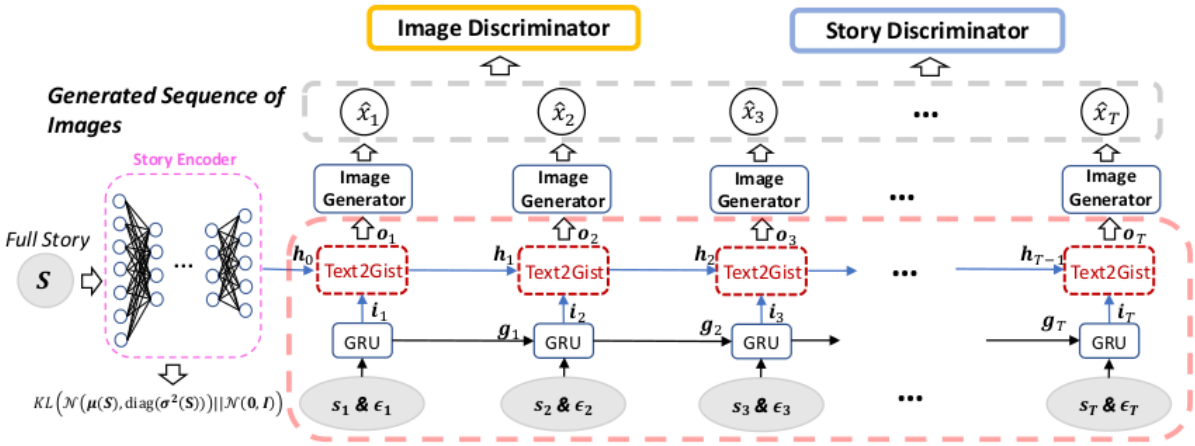
StoryGAN

GitHub Repo - https://github.com/yitong91/StoryGAN



StoryGAN Model Outline

Components-

i) **Story Encoder** - encodes story **S** into a low-dimensional vector $\mathbf{h}_0$

ii) **Context Encoder** - encodes input sentence $\mathbf{s}_t$ and its contextual information $\mathbf{o}_t$ (**Gist**) for each time point **t**

iii) **Image Generator -** generates image $\mathbf{x}_t$ based on $\mathbf{o}_t$ for each time step **t**

iv) **Image Discriminator & Story Discriminator -** guide the image generation process so as to ensure the generated image sequence **X** is locally and globally consistent, respectively.

Text2Gist-

The lower layer is implemented using standard GRU cells and the upper layer using the proposed Text2Gist cells, which are a variant to GRU cells and are detailed below. At

time step t, the GRU layer takes as input the concatenation of the sentence st and isometric Gaussian noise t , and outputs the vector it . The Text2Gist cell combines the GRU's output it with the story context h_t (initialized by Story Encoder) to generate o_t that encodes all necessary information for generating an image at time t. h_t is updated by the Text2Gist cell to reflect the change of potential context information.

How Does Context Encoder Work?

Let $\boldsymbol{g}_t$ and $\boldsymbol{h}_t$ denote the hidden vectors of the GRU and Text2Gist cells, respectively. The Context Encoder works in two steps to generate its output:

$$\boldsymbol{i}_t, \boldsymbol{g}_t = \text{GRU}(\boldsymbol{s}_t, \boldsymbol{\epsilon}_t, \boldsymbol{g}_{t-1}), \quad (2)$$

$$\boldsymbol{o}_t, \boldsymbol{h}_t = \text{Text2Gist}(\boldsymbol{i}_t, \boldsymbol{h}_{t-1}). \quad (3)$$

How is input fed in and modified?

We call $\boldsymbol{o}_t$ the *"Gist" vector* since it combines all the global and local context information, from $\boldsymbol{h}_{t-1}$ and $\boldsymbol{i}_t$ respectively, at time step t (*i.e.*, it captures the "gist" of the information). The Story Encoder initializes $\boldsymbol{h}_0$, while $\boldsymbol{g}_0$ is randomly sampled from an isometric Gaussian distribution.

We next give the underlying updates of Text2Gist. Given $\boldsymbol{h}_{t-1}$ and $\boldsymbol{i}_t$ at time step t, Text2Gist generates a hidden vector $\boldsymbol{h}_t$ and an output vector $\boldsymbol{o}_t$ as follows:

$$\boldsymbol{z}_t = \sigma_z\left(\boldsymbol{W}_z\boldsymbol{i}_t + \boldsymbol{U}_t\boldsymbol{h}_{t-1} + \boldsymbol{b}_z\right), \tag{4}$$

$$\boldsymbol{r}_t = \sigma_r\left(\boldsymbol{W}_r\boldsymbol{i}_t + \boldsymbol{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r\right), \tag{5}$$

$$\begin{aligned}\boldsymbol{h}_t &= (\mathbf{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} \\ &\quad + \boldsymbol{z}_t \odot \sigma_h\left(\boldsymbol{W}_h\boldsymbol{i}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h\right),\end{aligned} \tag{6}$$

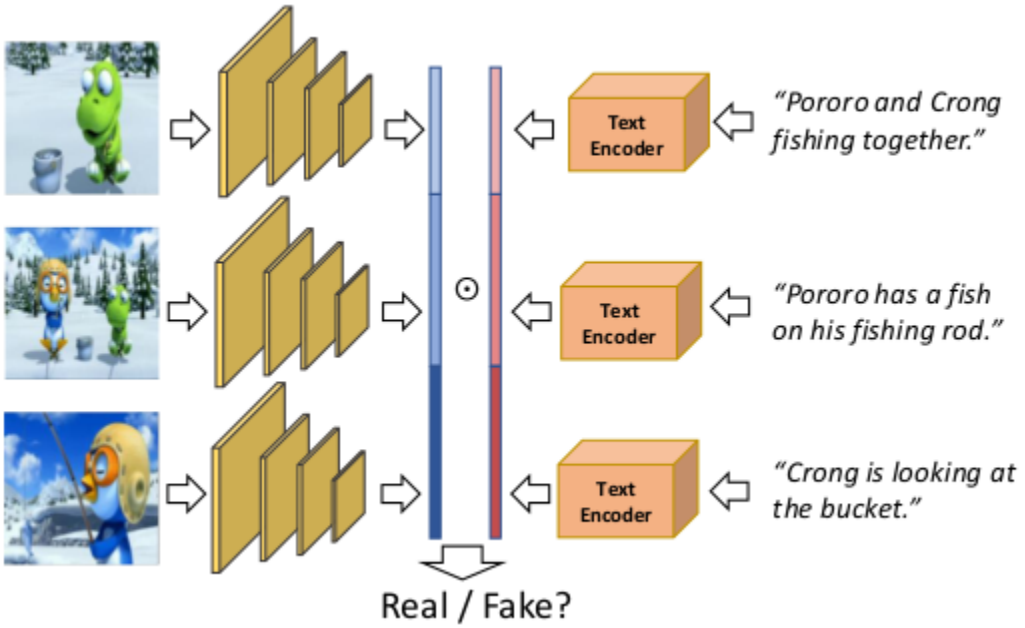
$$\boldsymbol{o}_t = \text{Filter}(\boldsymbol{i}_t) * \boldsymbol{h}_t, \tag{7}$$

where $\boldsymbol{z}_t$ and $\boldsymbol{r}_t$ are the outputs from the update and reset gates, respectively. The update gate decides how much information from the previous step should be kept, and the reset gate determines what to forget from $\boldsymbol{h}_{t-1}$. σ_z, σ_r and σ_h are sigmoid non-linearity functions. In contrast to standard GRU cells, output $\boldsymbol{o}_t$ is the convolution between Filter$(\boldsymbol{i}_t)$ and $\boldsymbol{h}_t$. The filter $\boldsymbol{i}_t$ is learned to adapt to $\boldsymbol{h}_t$. Specifically, Filter$(\cdot)$ transforms vector $\boldsymbol{i}_t$ to a multi-channel filter of size

$C_{out} \times 1 \times 1 \times \text{len}(\boldsymbol{h}_t)$ using a neural network, where C_{out} is the number of output channels. Since $\boldsymbol{h}_t$ is a vector, this filter is used as a 1D filter as in a standard convolutional layer.

The convolution operator in Eq. (7) infuses the global contextual information from $\boldsymbol{h}_t$ and local information from $\boldsymbol{i}_t$. $\boldsymbol{o}_t$ is the output of the Text2Gist cell at time step t. Since $\boldsymbol{i}_t$ encodes information from $\boldsymbol{s}_t$ and $\boldsymbol{h}_t$ from $\boldsymbol{S}$, which reflects the whole picture of the story, the convolutional operation in Eq. (7) can be seen as helping $\boldsymbol{s}_t$ to pick out the important part from the story in the process of generation. Empirically, we find that Text2Gist is more effective than traditional RNNs for story visualization.

How does Image and Story Discriminator Work?



Compares generated triplets $\{s_t, h_0, {}^*x_t\}$ (generated) and $\{s_t, h_0, x_t\}$ (ground truth)

The story discriminator helps enforce the global consistency of the generated image sequence given story S. It differs from the discriminators used for video generation, which often use 3D convolution to smooth the changes b/w the frames.

the changes between frames. The overall architecture of the story discriminator is illustrated in Figure 3. The left part is an image encoder, which encodes an image sequence into a sequence of feature vectors $E_{img}(\boldsymbol{X}) = [E_{img}(\boldsymbol{x}_1), \cdots, E_{img}(\boldsymbol{x}_T)]$, where $\boldsymbol{X}$ are either real or generated images (which are denoted by $\hat{\boldsymbol{X}}$). These vectors are concatenated into a single vector, shown as the blue rectangular in Fig. 3. Similarly, the right part is a text encoder, which encodes the multi-sentence story S into a sequence of feature vectors $E_{txt}(\boldsymbol{S}) = [E_{txt}(\boldsymbol{s}_1), \cdots, E_{txt}(\boldsymbol{s}_T)]$. Likewise, these are concatenated into one big vector, shown as the red rectangle in Fig. 3. The image encoder is implemented as a deep convolutional network and the text encoder as a multi-layer perceptron. Both output a same dimensional vector.

The global consistency score is computed as

$$D_S = \sigma\left(\boldsymbol{w}^\intercal\left(E_{img}(\boldsymbol{X}) \odot E_{txt}(\boldsymbol{S})\right) + b\right), \qquad (8)$$

Datasets

i) CLEVR-SV

ii) Cartoon Dataset (Pororo)

About 40 video clips forms a complete story. Each story has several QA pairs. In total, the Pororo dataset contains 16K clips of one second videos about 13 distinct characters. The
manually written description has an average length of 13.6 words that describes what is happening and which characters are in each video clip. These 16K video clips are sorted
into 408 movie stories.

We modified the Pororo dataset to fit story visualization task by considering the description for each video clip as the story's text input. For each video clip, we randomly pick out one frame (sampling rate is 30Hz) during training as the real image sample. Five continuous images form a single story. Finally, we end up with 15, 336 description-story pairs, where 13, 000 pairs are used as training, the remaining 2, 336 pairs for testing. We call this dataset Pororo-SV to differ it from the original Pororo QA dataset.