Implementation details of creative A.I models

Creative A.I has widely been in use for many advertising and marketing strategies in Russia and all over the world since the invention of GANs (generative adversarial nets) for data synthesis (Goodfellow et al, 2014). Many companies and industries are extensively using Creative A.I models to generate attractive content such as images, videos for attractive advertising of products and services. Apart from applications of Creative A.I in marketing and strategies, it is widely been in use by many social media companies such as Facebook, Twitter, Instagram for better user experience. Generating brand new content using neural networks comes under multiple aspects depending on the context of advertising a product or service. In this document, I am going to mention the implementation details such as model architecture, algorithm, and maths behind applications that generate images and videos from input text. The exact implementation details can also be adapted with fine changes to perform tasks such as image completion, video completion, video generation from static image, etc. A.I applications such as text-to-image generation, text-to-video generation, image-to-speech generation can also help impaired people to understand the world better and faster. These state of art applications can improve the lives of all impaired people and also contribute to faster growth of businesses.

The Fine Tuning Process:
The implementation details mentioned below for text-to-image generation are also applicable for any kind of creative A.I models such as image completion, video completion, video generation from static image, text-to-image generation, text-to-video generation, image-to-speech generation. The applications that deal with video generation, video completion or video representation need additional libraries to process before representing them to the neural networks for training.

Processing videos for neural networks:
Representing unstructured data, especially image/video data to neural networks is a tough task. Inventions such as convolutional neural networks have worked well on representing image data to neural networks. In the pre-deep learning era, image data is represented through fully connected layers which could lead to loss of information of pixel position. Convolutional neural networks worked well in representing image data more appropriately to neural networks that enabled building more generalizability. For image data 2D convolutional layers are applied to represent the data to neural networks but for videos unlike images, they can't be represented using 2D convolution because of the fact that they have an extra temporal dimension i.e., frames as depth in this case. The exact working of 3D convolution is explained here. The important preprocessing steps before representing a video to neural networks to carrry out tasks related to video understanding or generation are:

1. Process video into frames i.e, a volume of 3D data
2. Apply 3D convolutions, 3D poolings over 3D data i.e., video.

The step one, processing video into frames is done using the python library open-cv with the code below:

```
import cv2

# Function to extract frames of a video
def Capture_frame(path):

    # Path/address of the video file
    video_obj = cv2.VideoCapture(path)

    # Used as counter variable
    counter = 0

    # variable to check weather frame has been extracted from the video
    is_frame_extracted = 1

    while is_frame_extracted :

        # video_obj object calls read
        # function extract frames
        is_frame_extracted , image = video_obj .read()

        # Saves the frames with frame-count
        cv2.imwrite("frame%d.jpg" % counter , image)

        counter += 1
# Call the function Capture_frame with the path of any video to extract the
frames.
```

Then step two is to apply 3D convolution over the frames. Deep learning frameworks such as tensorflow and pytorch provides functions to implement 3D convolution on video data. Example code to implement 3D convolution is below (The part of code is taken form the official documentation of keras here and tensorflow check this):

```
# create a sequential object
model = Sequential()
# apply a 3D convolutional operation over frames of video
model.add(Convolution3D(filters,kernel_size, strides, padding, input_shape=(1,
img_rows, img_cols, patch_size), activation='relu'))
# apply a 3D max pool layer
model.add(MaxPooling3D(pool_size=pool_size))
# apply a drop out layer
model.add(Dropout(0.5))
# flatten the output
model.add(Flatten())
# Note: Add the similar set of layers for more then 3 times to start training
#excluding input_shape in Conv3D layer
```

Note: Lot of ideas, concepts, code examples are taken from paper of Scott R, Zeynep A, Xinchen Y, Lajanugen L, Bernt, Lee H. “Generative Adversarial Text to Image Synthesis”. arXiv:1605.05396, 2016. The code examples are taken from paperswithcode.com in GitHub repository on above paper..

Architecture:

1. Generative Adversarial Networks (Goodfellow et al, 2014):
   Generative adversarial networks (GANs) consist of a generator G and a discriminator D that compete with each other in such a way that the discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator by producing much more enhanced and realistic data. The improved version of GAN is used in the applications of text-to-image. The actual end-to-end architecture is discussed below.
   The GAN’s loss function as shown in figure 1 is used to train models such that the data produced by generator G is too realistic to distinguish the generated and original data.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{x\sim p_z(z)}[\log(1 - D(G(z)))]$$

Figure1

The GAN model is found to learn better and faster in practice for the generator to maximize log(D(G(z))) instead of minimizing log(1 − D(G(z))).

The visual architecture of GAN consisting of generator G, discriminator D is in figure2.

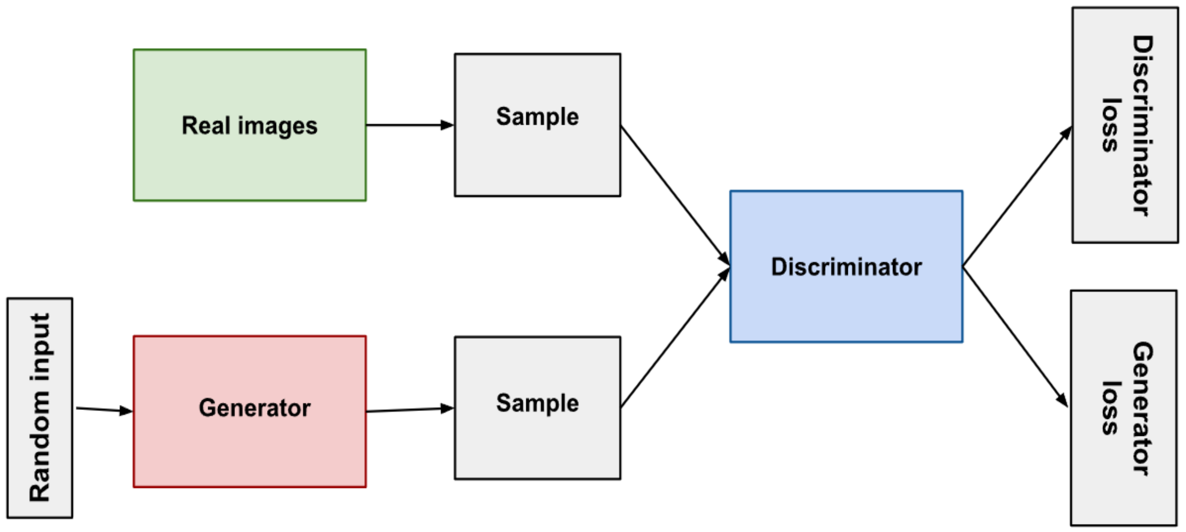


Figure2

2. **Deep Convolutional Recurrent Neural Network** (Reed et al, 2016):
Deep Convolutional Recurrent Neural Network consists of both convolutional and sequential models to encode the text descriptions. Deep convolutional recurrent neural networks are used because of the fact that convolutional neural networks are fast and scale well on long sequences whereas recurrent neural networks are good at efficiently learning the temporal low-level features. The encoder network of deep convolutional recurrent neural nets can learn the correspondence function with the image labels associated with the text description in the training set. The network architecture is in such a way that the CNN has ReLU activation to split along the time dimension and treated as input sequence of vectors to recurrent neural networks i.e., LSTM (Long Short Term Memory) in this case. The entire resulting network is still end-to-end differentiable. This idea of encoding characters using deep convolutional recurrent neural networks also works for encoding words. The loss function used for optimising the encoder network for text characters is in figure3, figure4 (e Akata et al, 2015).

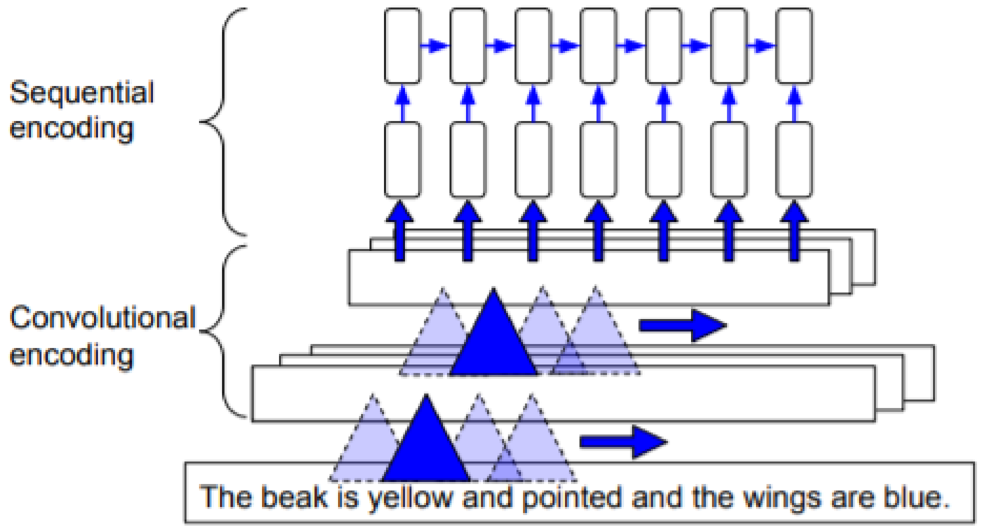
$$\frac{1}{N}\sum_{n=1}^{N}\Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

Figure3

$$f_v(v) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{t\sim\mathcal{T}(y)}[\phi(v)^T\varphi(t))]$$
$$f_t(t) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{v\sim\mathcal{V}(y)}[\phi(v)^T\varphi(t))]$$

Figure4

The inference of formulae in figure3, figure4 is from the paper (Akata et al, 2015).
The visual architecture of deep convolutional recurrent neural networks which is applied in the dimension of time along the activation is as shown in figure5.



3. **The algorithm of text-to-image generation:**
   The figure7 explains the step by step algorithm of implementing the text-to-image generation with methods and formulas mentioned above.

Algorithm 1 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

```
 1: Input: minibatch images $x$, matching text $t$, mis-matching $\hat{t}$, number of training batch steps $S$
 2: for $n = 1$ to $S$ do
 3:     $h \leftarrow \varphi(t)$ {Encode matching text description}
 4:     $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
 5:     $z \sim \mathcal{N}(0,1)^Z$ {Draw sample of random noise}
 6:     $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
 7:     $s_r \leftarrow D(x, h)$ {real image, right text}
 8:     $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
 9:     $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
10:     $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
11:     $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
12:     $\mathcal{L}_G \leftarrow \log(s_f)$
13:     $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
14: end for
```

Figure7

4. **End-on-End Architecture of text-to-image generation combining GANs and Deep Convolutional Recurrent Neural Network:**
   The formula used to learn the style to transfer style is in figure8 below:

$$\mathbb{E}_{t_1, t_2 \sim p_{data}} [\log(1 - D(G(z, \beta t_1 + (1 - \beta) t_2)))]$$

Figure8

The formula used to learn the interpolation is in the formula9 below:

$$\mathcal{L}_{style} = \mathbb{E}_{t, z \sim \mathcal{N}(0,1)} ||z - S(G(z, \varphi(t)))||_2^2$$

Figure9

Further intuitive details behind the formulae are explained original paper mentioned in the beginning of the document.

The formula10 below shows the architecture that performs very well on a wide range of generative applications.
Note: Figure10 below is not the original architecture discussed in the paper on text-to-image generation. The architecture used in the paper is here.

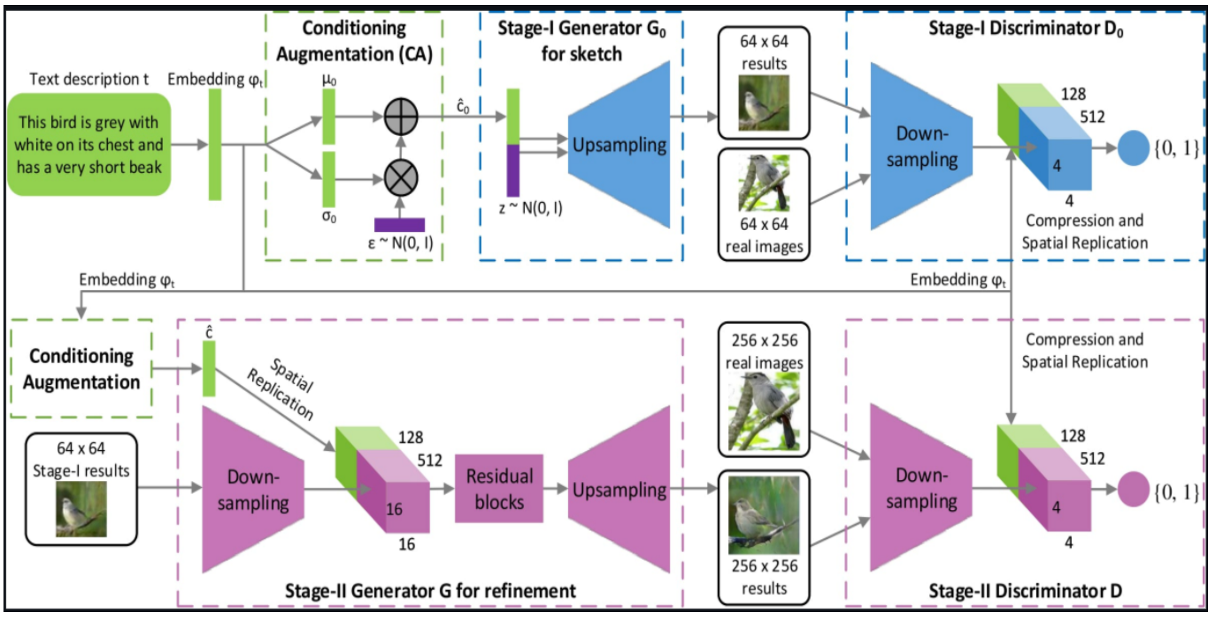


Figure10