Hi everyone, my name is Juncen. I represent our group doing this presentation.

First, let me briefly introduce the main idea, which differs from the mainstream approach (/əˈproʊtʃ/) to implementing (ˈɪmpləmentɪŋ) the Image Style transfer.

As we know, Generative Adversarial (/ˌædvərˈseriəl/) Network (GAN) is a widely adopted framework for this task because of its better representation ability of local style patterns compared with the traditional (/trəˈdɪʃən(ə)l/) Gram-matrix-based methods.

However, most previous methods rely on sufficient pre-collected style images to train the model. Our approach proposes a novel Patch Permutation GAN (P2-GAN) network that can efficiently learn the stroke (/stroʊk/) style from a single image.

We use patch permutation to generate multiple training samples from the given (/ˈɡɪv(ə)n/
) style image. A patch discriminator is designed to process patch-wise and natural photos simultaneously (/ˌsaɪm(ə)lˈteɪniəsli/).

We also propose a local texture (/ˈtekstʃər/) descriptor-based criterion (/kraɪˈtɪriən/) to evaluate the style transfer quality. Experimental results showed that our method could produce finer-quality results from single-style images with improved computational efficiency compared with other approaches.

And offline learning methods always use the feed (/fiːd/) -forward (/ˈfɔːrwərd/) networks, mostly auto-encoder, trained from pre-collected style images to memorize the style information so that the target image can be obtained (/əbˈteɪnd/) in real-time through feed-forward computation. These methods can be regarded as a trade of computational time from the online stage to the offline stage.

In this project, we are interested in the offline learning-based real-time style transfer method and propose to use only one style image for the training of GANs. It will make it possible to precisely simulate the typical (/ˈtɪpɪk(ə)l/) stroke style from the customer-designated image without being interfered (/ˌɪntərˈfɪr/) with by other sources. Through Patch Permutation GAN (P2-GAN) network, we aim to randomly break the unique style image into multiple patches used as the training set. The structure of content images is preserved through an encoder-decoder generator.

Network Architecture:

When M = 1 (the number of style images), the training stage of standard GANs will become unstable since a single sample cannot represent a distribution, and the network may even collapse while training. As shown in this slide, we have designed a novel network architecture to overcome this problem. A patch permutation module ψ (·) and a patch discriminator Dp (·) are proposed. Together with an improved encoder-decoder generator G (·) for more efficient computations on both offline and online stages, high-quality stroke style transfer using single-style images can be accomplished in real-time.

Patch Permutation:

Since we only have one style image, we consider breaking the style image into multiple patches and learning stroke style from the patches. Therefore, a permutation method for the style image on the picture is proposed.

Path Discriminator:

We also adopt the L-layer CNN structure similar to PatchGAN. However, unlike PatchGAN's setup, where a relatively small stride (1 or 2) is applied, we restrict a special relationship between the stride and other network parameters.

Encoder-Decoder Generator:

To address the problem of checkerboard artifacts incurred by the traditional deconvolution (usually as transposed convolution), we adopted the resize convolution operation before the DSC in each decoder layer. Using the nearest-neighbor interpolation, the feature maps will be upsampled into two times longer than their original size. In all encoder layers and decoder layers, relu is used as the activation (/ˌæktɪˈveɪʃn/) function.

comparison /kəmˈpærɪsn/

Results:

We also compare the earliest way proposed by Gatys and our Patch Permutation GAN. The results are shown in this slide. Also, we implement using Tensorflow. We wrote the readme file about the details. Follow the readme file if you are willing to run our codes in your environments.