AnimeGAN: A Novel Lightweight GAN for Photo Animation

Jie Chen · Gang Liu · Xin Chen

Springer, Singapore
DOI: 10.1007/978-981-15-5577-0_18
First online: 26 May 2020

Introduction:

A novel approach for transforming photos of real-world scenes into anime style images is proposed.

-Not only allows the artists to focus on more creative work and save time, but also makes it easier for ordinary people to implement their own animation.

-Generative adversarial networks (GANs): 以假乱真

Generative（生成）：GAN实际上可以看作是一个生成数据的工具。目标就是通过学习让自身生成更加真实的数据。

Adversarial （对抗）：既然能够以假乱真，对抗的自然就是识别真假的警察了。

生成器（Generator ）：生成器努力使生成的图像更加真实，以骗过判别器。

判别器（Discriminator）：判别器则需要努力对图片进行真假判别，以识别出真假。

Problems: 1) the generated images have no obvious animated style textures; 2) the generated images lose the content of the original photos; 3) a large number of the parameters of the network require more memory capacity.

Data and method:

·Grayscale style loss: affects the animation transformation process in the generator G.

·Grayscale adversarial loss: helps to make the generated image retain the content of the input photo.

·Color reconstruction loss: makes the generated images have the clear anime style on the textures and lines.

$$L(G,D) = \omega_{adv} L_{adv}(G,D) + \omega_{con} L_{con}(G,D) + \omega_{gra} L_{gra}(G,D) + \omega_{col} L_{col}(G,L$$

· 6656 real-world photos are employed for training.

· set to 256 x 256

The Wind Rises (Miyazaki Hayao), 1792 animation images;

Your Name (Makoto Shinkai), 1650 animation images;

Paprika (Kon Satoshi), 1553 animation images.

Results:

Table 1. Comparisons of the performance of the different generators

Network	Params	Model size	FLOPs	Inference time
CartoonGAN	12253152	46.74M	108.98B	51 ms/image
AnimeGAN	**3956096**	**15.09M**	**38.66B**	**43 ms/image**

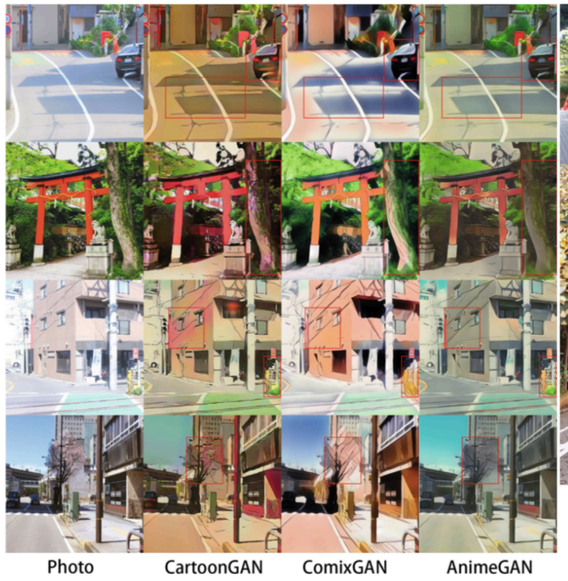


Fig. 3. Qualitative comparison of CartoonGAN, ComixGAN and AnimeGAN.

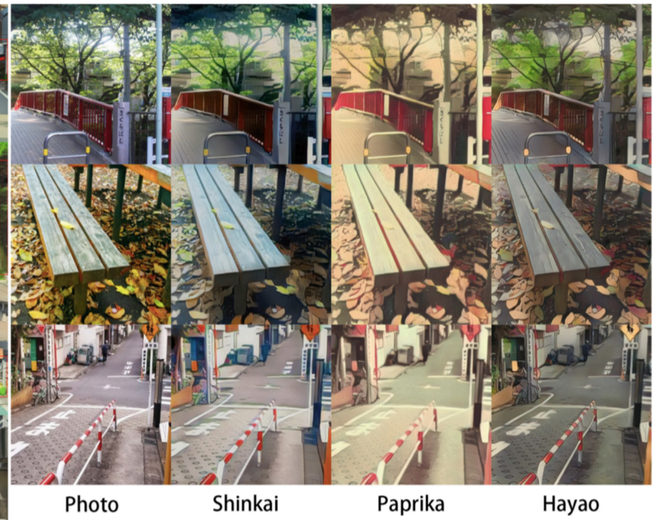


Fig. 6. The results produced by AnimeGAN for 3 different styles.

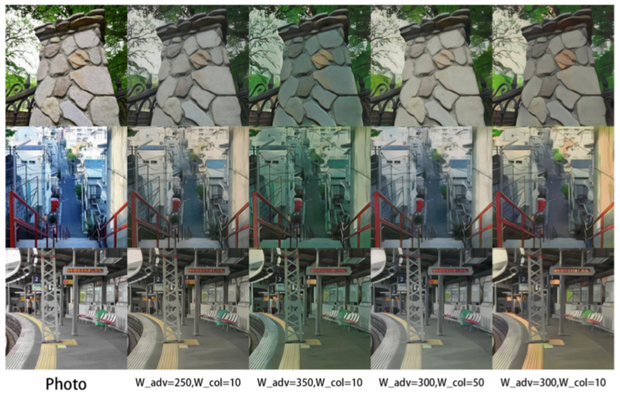
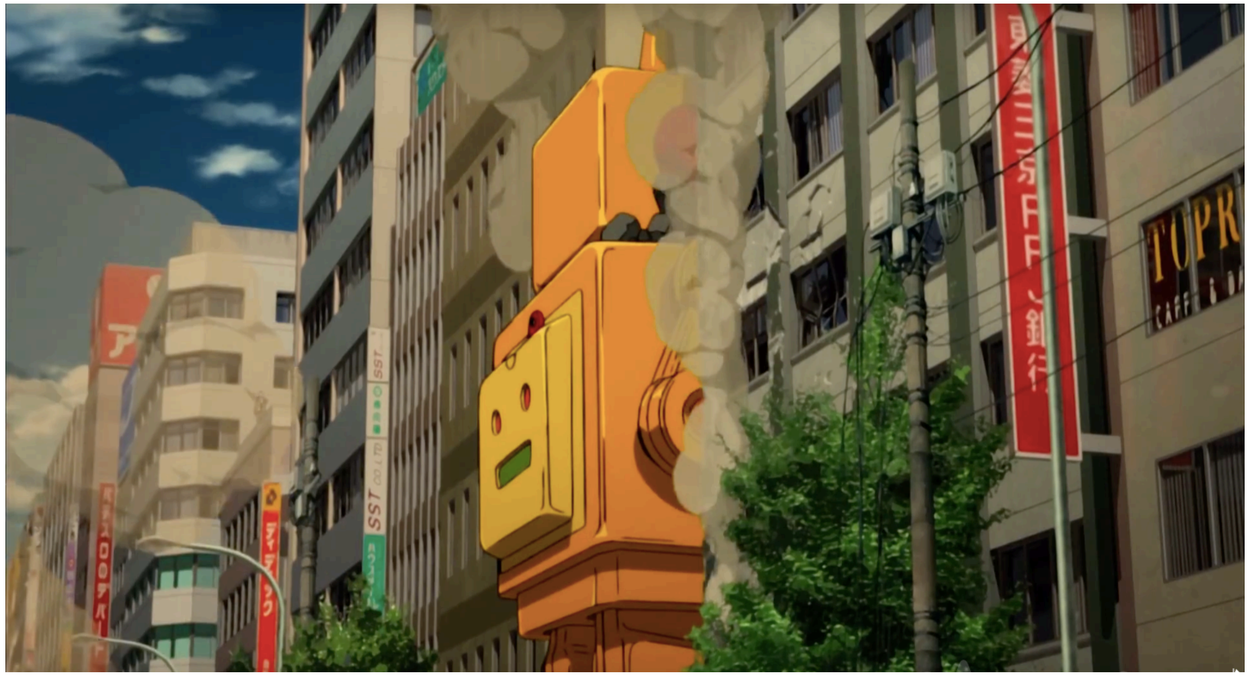


Fig. 4. Qualitative comparison of different weights. (Color figure online)

“用成百上千大触画师的作品训练出能让成百上千大触画师吃不上饭的模型。”

2020/06/15



Report

2020.06.18

張慕琪

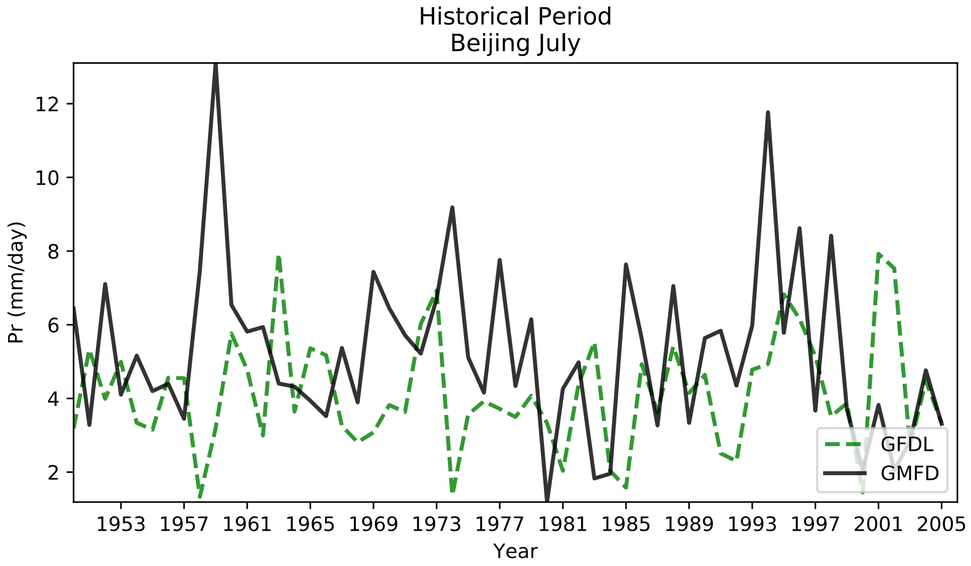
讨论

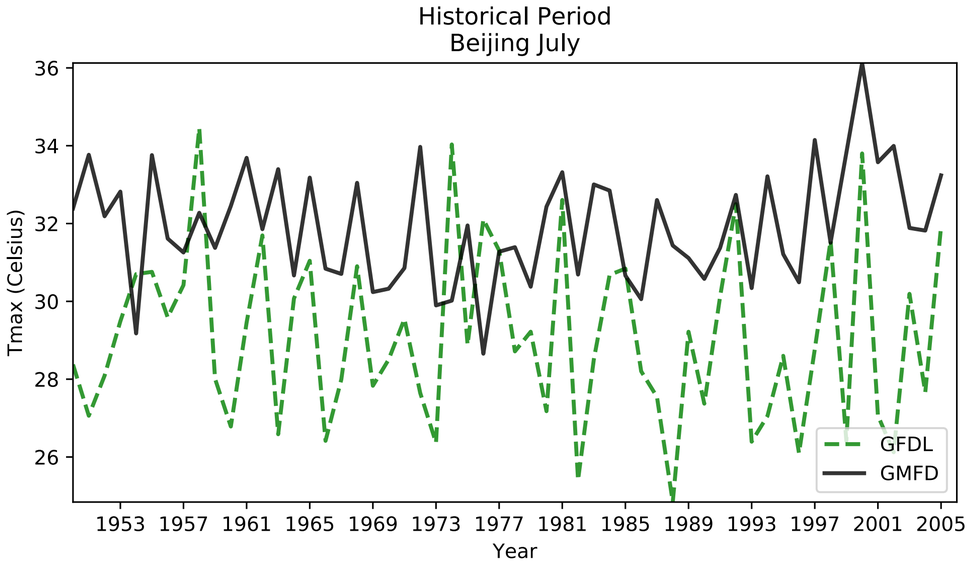
- 先开始GFDL的Tmax & Tmin和Pr训练；
- 关于Tmax&Tmin的ANN：

MLPRegressor：

hidden_layer_sizes=(10, 5),
max_iter=100,
tol=1e-3,
solver='sgd',
activation='relu',
learning_rate='adaptive',
learning_rate_init=1e-3,
alpha=1e-2

讨论


Historical Period
Beijing July
Pr (mm/day)
Year
GFDL
GMFD
2
4
6
8
10
12
1953
1957
1961
1965
1969
1973
1977
1981
1985
1989
1993
1997
2001
2005


Historical Period
Beijing July
Tmax (Celsius)
Year
GFDL
GMFD
26
28
30
32
34
36
1953
1957
1961
1965
1969
1973
1977
1981
1985
1989
1993
1997
2001
2005

讨论

activation{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default=’relu’
Activation function for the hidden layer.

- ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x
- ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
- ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)

solver{‘lbfgs’, ‘sgd’, ‘adam’}, default=’adam’
The solver for weight optimization.

- ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
- ‘sgd’ refers to stochastic gradient descent.
- ‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, ‘lbfgs’ can converge faster and perform better.

讨论

alpha, default=0.0001
L2 penalty (regularization term) parameter.
batch_sizeint, default=’auto’
Size of minibatches for stochastic optimizers. If the solver is ‘lbfgs’, the classifier will not use minibatch. When set to “auto”, batch_size=min(200, n_samples)

learning_rate{‘constant’, ‘invscaling’, ‘adaptive’}, default=’constant’
Learning rate schedule for weight updates.

- ‘constant’ is a constant learning rate given by ‘learning_rate_init’.

‘invscaling’ gradually decreases the learning rate learning_rate_ at each time step ‘t’ using an inverse scaling exponent of ‘power_t’. effective_learning_rate = learning_rate_init / pow(t, power_t)

- ‘adaptive’ keeps the learning rate constant to ‘learning_rate_init’ as long as training loss keeps decreasing. Each time two consecutive epochs fail to decrease training loss by at least tol, or fail to increase validation score by at least tol if ‘early_stopping’ is on, the current learning rate is divided by 5.

Only used when solver=’sgd’.

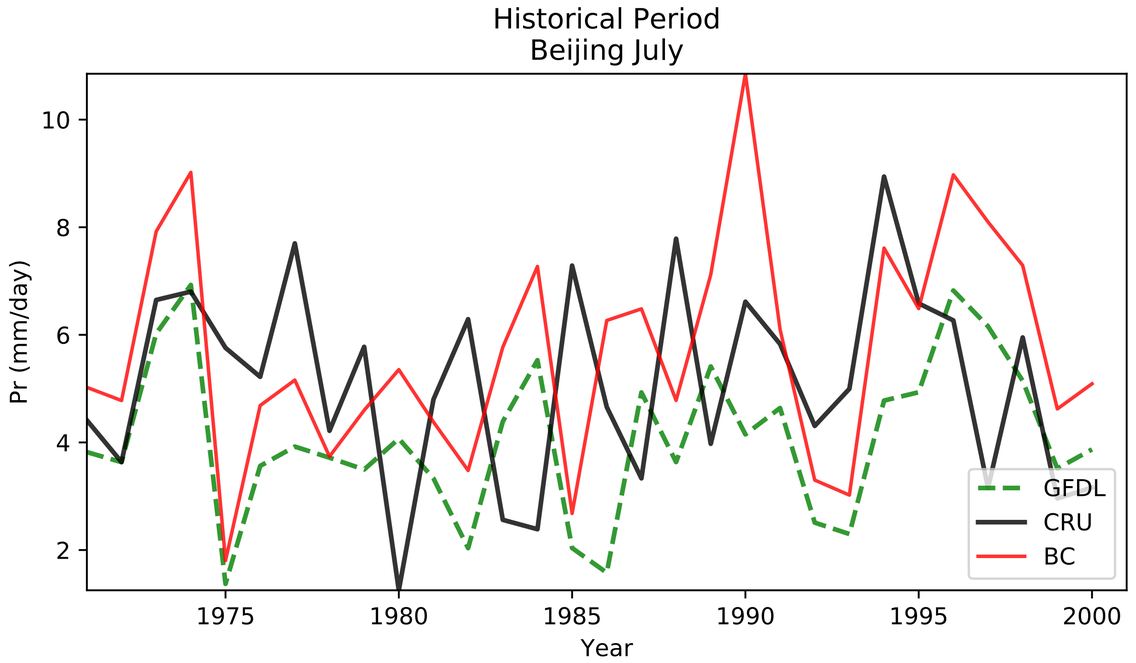
learning_rate_init, default=0.001
The initial learning rate used. It controls the step-size in updating the weights. Only used when solver=’sgd’ or ‘adam’.

讨论

max_iter, default=200
Maximum number of iterations. The solver iterates until convergence (determined by ‘tol’) or this number of iterations. For stochastic solvers (‘sgd’, ‘adam’), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

tol, *default=1e-4*
Tolerance for the optimization. When the loss or score is not improving by at least tol for n_iter_no_change consecutive iterations, unless learning_rate is set to ‘adaptive’, convergence is considered to be reached and training stops.

讨论




```
rmse_gfdl = 2.528
rmse_bc   = 2.638

cc_gfdl   = -0.007
cc_bc     = 0.117
```

谢谢