Progressive growing of GANs

(Progresszív növekvő GAN)

Váli Valter Nandrean David Cristian Csató Erik

Abstract

We tried to generate synthetic images of humans with Generative adversarial networks (also known as GANs) using the CelebA-HQ_256 dataset. GANs are trained with two sub-models: the generator that tries to generate fake data and the discriminator that tries to classify the data as fake or real. These two are trained together until the discriminator is fooled enough times. In our attempt of synthetizing fake human images we used a subtype of GANs, the progressive growing generative adversarial networks. These networks use an updated version of the classic training method that generates small images at the beginning of the training process and they incrementally increase the size until they reach the target output size. The network architecture was implemented in the Python programming language using Tensorflow and Keras. Our goal was to achieve the image size of 256×256. For the training process we used the Azure Machine Learning environment because we did not have the appropriate hardware at our disposition. We validate our results with good quality generated images.

Kivonat

Feladatunk generatív adverzális hálózatokkal (más néven GAN-okkal) szintetikus emberi képek generálása volt a CelebA-HQ_256 adathalmaz segítségével. A GAN-ok két fő részből állnak: a generátorból, amely hamis adatokat próbál létrehozni, és a diszkriminátorból, amely megpróbálja a hamis és valódi adatokat megkülönböztetni egymástól. Ezen két hálózat tanítását együtt végezzük egészen addig, amíg a generator olyan adatokat nem general, amiket a diszkriminátor már nem képes megkülönböztetni a valódi adatoktól. A hamis emberi arcok generálásakor a GAN-ok egy típusát, a progresszív növekvő generatív adverzális hálózatot használtuk. Ezek a hálózatok a klasszikus módszer egy frissített változatát használják. A tanítási folyamat elején kis méretű képeket generálunk, és fokozatosan növeljük a képek felbontását, amíg el nem érjük a kívánt méretet. A hálózatunkat Python programozási nyelven, a Tensorflow és Keras könyvátrak segítségével valósítottuk meg. Célunk a 256×256-os méret elérése volt. A tanításhoz az Azure Machine Learning környezetet használtuk, mivel nem állt rendelkezésünkre megfelelő hardver. Eredményeinket jó minőségű generált képekkel validáltuk.

I. Preprocessing

For preprocessing we loaded the images into an array using the numpy package and cropped the faces out of them using the MTCNN face detection model, after that we resized the images to the desired output size. We compressed and saved these 256×256 cropped faces into a single .npz file and uploaded them to Azure. At the beginning of the training process we load the compressed images and normalize them. Before upgrading the output size in the training process we also resize the images.

II. Azure machine learning

We used Azure Machine Learning to train the progressive growing GAN. The network was trained on the Standard_NC6 virtual machine, with 6 cores, 56 GB RAM and 380 GB disk size. The processing unit was an NVIDIA Tesla K80 GPU. The dependencies and required packages were specified with a .yml file according to the Azure documentation. Every train was launched from a local computer using the azureml-core python package. The final training process lasted 19 hours and 23 minutes.

III. Network

Our project implements the same architecture that was given in the PGGAN paper. In the first phase, it takes a latent vector and uses two 2D convolution layers to generate 4×4 images. Then we train the discriminator with the generated images and the 4×4 real images. Once the training stables, we add two more 2D convolution layers to the generator to upsample the image to 8×8 and 2 more convolution layers to downsample images in the discriminator. When doubling the resolution of the generator and discriminator, we fade in the new layers smoothly with weight α gradually increases linearly from 0 to 1 below. After 7 of these phases we get the 256x256 pictures. This progressive training speeds up and stabilizes the regular GAN training methods.

We took an article (https://machinelearningmastery.com/how-to-implement-progressive-growing-gan-models-in-keras/) as a basis for the architecture but made some expansion and modification on it.

IV. Hyperparameter Tuning

Since we build the same architecture, which was implemented in the PGGAN paper the hyperparameters - such as desired batch size for every resolution, best optimizer, loss function etc. - were given. During the research phase we read some papers about GANs but we did not find any methods for hyperparameter tuning nor its necessity is mentioned in them. The best advice we found was that the hyperparameters of already implemented architectures are described in their papers. A possible reason for this can be the strong volatility and the use of other hyperparameter of the GAN training, which will lead to mode collapse or vanishing gradients. Hence we took over those parameters to our implementation.

Nevertheless, the hyperparameter optimization is possible but pretty expensive, especially for GANs. The reason for that is that the only way to evaluate the hyperparameter optimization results for GANs is to compare the generated images with each other.
If we had more time and computational resources maybe changing the learning rate (of the optimizer) or the batch sizes of every resolution we would have achieved some improvement, but this process -even with powerful GPUs- would have last days in such a complex architecture we have.

V. Training

We tried two different approaches. Firstly we used binary cross entropy as loss function and batch normalization in each convolutional layer (with 0.8 momentum rate) on the generator side. It worked promising until resolution 16x16 but the 32x32 output images were too similar because of the mode collapse. The loss function of the second approach was MSE, furthermore, we took out the batch normalization layers of the network. This combination worked stable so we decided on this approach.

Following we suggest some observations which could definitely advance the image generation.

1) The usage of a simplified Minibatch discrimination on the discriminator side would have also improved the image diversity.

2) To generate better and more realistic images we could have used twice as much images or simply data augmentation.

3) Maybe our solution could be better with pixelnormalization in the convolutional layers (in discriminator and generator) and with wasserstein loss.

VI. Testing

We didn't split the dataset into train, validation, and test sets, instead we used all 30000 images to train our network because generally GANs cannot be tested the same way as other neural networks. Therefore, we tested our GAN by generating fake images and deciding whether they look realistic enough or not.

VII. Android application

We created an Android application to demonstrate the capabilities of our generator model, where you can set the desired resolution and then generate realistic images of non-existent people.

References

[1] GAN: https://arxiv.org/pdf/1710.07035.pdf

[2] GAN: https://arxiv.org/pdf/1406.2661.pdf

[3] GAN: https://dl.acm.org/doi/abs/10.1145/3422622

[4] PGGAN: https://arxiv.org/pdf/1710.10196.pdf

[5] PGGAN: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8751969

[6] Wasserstein: https://arxiv.org/abs/1506.05439

[7] Resolution: http://proceedings.mlr.press/v70/odena17a/odena17a.pdf

[8] Convolutional neural network: https://arxiv.org/pdf/1404.2188v1.pdf

[9] hyperparameters tuning: https://arxiv.org/pdf/2003.05689.pdf

[10] MTCNN : https://arxiv.org/ftp/arxiv/papers/1604/1604.02878.pdf