W5 GAN道しるべ

Competition page:

<https://www.kaggle.com/competitions/gan-getting-started>

**GAN Implementation tips**

* Scale the image pixel value between -1 and 1. Use tanh as the output layer for the generator.
* Experiment sampling ***z*** with Gaussian distributions.
* Batch normalization often stabilizes training.
* Use PixelShuffle and transpose convolution for upsampling.
* Avoid max pooling for downsampling. Use convolution stride.
* Adam optimizer usually works better than other methods.
* Add noise to the real and generated images before feeding them into the discriminator.

<https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b>

MNISTのGANｽｸﾗｯﾁ実装

https://qiita.com/mgmk2/items/4847e01501a3d56a3094

**ｻﾗﾐ：ペア画像必要なし！教師なし学習で光学画像とSAR画像を相互変換する**

<https://sorabatake.jp/15701/>

この資料↓超読み易い！写経おすすめ

Keras-GAN-cycleGAN

<https://github.com/eriklindernoren/Keras-GAN/blob/master/cyclegan/cyclegan.py>

Keras-GAN-pix2pix

<https://github.com/eriklindernoren/Keras-GAN/blob/master/pix2pix/pix2pix.py>

W5PeerReview；

**I’m Something of a Painter Myself**

<https://www.kaggle.com/competitions/gan-getting-started>

参照；

Pytorch code with bronze medal：

<https://www.kaggle.com/code/chuyangzhang/i-m-something-of-a-painter-myself-cyclegan>

Monet CycleGAN Tutorial by Amy Jang

<https://www.kaggle.com/code/amyjang/monet-cyclegan-tutorial>

実装練習；

DCGAN implementation by Keras

<https://keras.io/examples/generative/dcgan_overriding_train_step/>

<https://www.tensorflow.org/tutorials/generative/dcgan>

Using tf.GradientTape

<https://www.tensorflow.org/guide/autodiff>

# Introduction

Welcome!

このノートブックでは、偉大な巨匠モネの絵画を写真調に、また風景写真をモネの絵画調に変換するGenerative Adversarial Network, cycleGANの仕組みをｺｰﾄﾞを通して探索します。

また私自身、TensorFlowでdeep learningの本格的なｺｰﾄﾞに取り組むのは初めてです。そのためこのNotebookでは、GANの構造の探索に留まらず、TensorFlowにおけるDeep Learningのｺｰﾃﾞｨﾝｸﾞの構造を並行して探索します。

TensorFlowに慣れている方には、既知の情報がしつこく記述されて恐縮ですが、TensorFlowに慣れていない方には有用な情報が少しでもお役に立てば幸いです。

以下、目次：

# Discriminator

The discriminator is a neural network trained to distinguish between real images and from the dataset and fake images generated by the generator.

# The patchGAN

patchGANは、画像全体ではなく、ﾛｰｶﾙな小さな領域ごとに判定を行います。例えば、256x256の画像に対して全範囲にfakeかrealかの判定を行うのではなく、例えば70x70の小さなﾊﾟｯﾁ単位で判断を行います。ﾊﾟｯﾁとは受容野のことです。この局所的な判定により高周波数成分（ｴｯｼﾞやﾃｷｽﾁｬなど細部の特徴）を捉えることができ、細部のﾘｱﾘﾃｨを向上させる効果があります。最終的に全ﾊﾟｯﾁの平均を計算して判定に利用します。この手法をpatchGANと呼びます。

# Three times downsampling

今回のdiscriminatorは3回のdownsamplingによって次元削減した特徴ﾏｯﾌﾟに調整を加えて、最終的に30x30のｽｺｱﾏｯﾌﾟを出力します。

ここで、30x30はｽｺｱﾏｯﾌﾟであり、ﾊﾟｯﾁｻｲｽﾞではないことにご注意ください。ﾊﾟｯﾁとは、30x30のｽｺｱﾏｯﾌﾟの「各ｾﾙ」が見ている、元の画像のﾋﾟｸｾﾙ領域（受容野）です。

今回のﾊﾟｯﾁｻｲｽﾞは70x70で、この限定された範囲からﾀﾞｳﾝｻﾝﾌﾟﾘﾝｸﾞ（通常はｽﾄﾗｲﾄﾞ付き畳み込み）によって特徴を抽出し、最終的に1つのｽｺｱ（確率）が付与されます。

30x30のｽｺｱﾏｯﾌﾟとは、個々のｾﾙに、そのｾﾙの受容野に対するｽｺｱが割り当てられたものです。

また、画像ｻｲｽﾞがより大きい場合や小さい場合は、適切なスコアﾏｯﾌﾟを出力するためdownsamplingの回数を調整します。例えば、より大きな画像ならﾀﾞｳﾝｻﾝﾌﾟﾘﾝｸﾞの回数を増やし、小さな画像なら減らします。

# Identity Mapping (Identity loss)

ここでもう一つの損失であるidentity lossを追加します。この損失は、生成画像と元の画像の誤差を測定することで、Generatorが画像の色調を維持することを学習します。

identity lossが無い場合、Generatorは元の画像に似たものを生成できたとしても、空が赤かったり、海が黄色だったりと、色調を上手く再現できないことがあります。

これを防ぐためにidentity lossが有効です。

identity lossは次の式で表されます：

$$

\mathcal{L}\_{identity}(G, F) =

\mathbb{E}\_{x \sim P\_{\text{data}}(x)}

\left[ ||F(x) - x||\_1 \right] +

\mathbb{E}\_{y \sim P\_{\text{data}}(y)}

\left[ ||G(y) - y||\_1 \right]

$$

LAMBDA = 12

Epoch 5/5

300/300 [==============================] - 79s 250ms/step - monet\_gen\_loss: 3.8285 - photo\_gen\_loss: 4.0929 - monet\_disc\_loss: 0.6338 - photo\_disc\_loss: 0.5853

LAMBDA = 10

Epoch 5/5

300/300 [==============================] - 145s 475ms/step - monet\_gen\_loss: 3.2847 - photo\_gen\_loss: 3.3653 - monet\_disc\_loss: 0.6294 - photo\_disc\_loss: 0.6132

LAMBDA = 8

Epoch 5/5

300/300 [==============================] - 74s 242ms/step - monet\_gen\_loss: 2.8941 - photo\_gen\_loss: 2.9720 - monet\_disc\_loss: 0.6323 - photo\_disc\_loss: 0.6246