Name: Raghav Chugh

Email address: chughraghav@gmail.com

Contact number: +91-9729485880

Anydesk address:

Years of Work Experience: **1.5**

Date: **05th Sep 2021**

Self Case Study -2: Image to Image Translation using cGAN

"After you have completed the document, please submit it in the classroom in the pdf format."

Please check this video before you get started:
https://www.youtube.com/watch?time_continue=1&v=LBGU1_JO3kg

Overview

Introduction

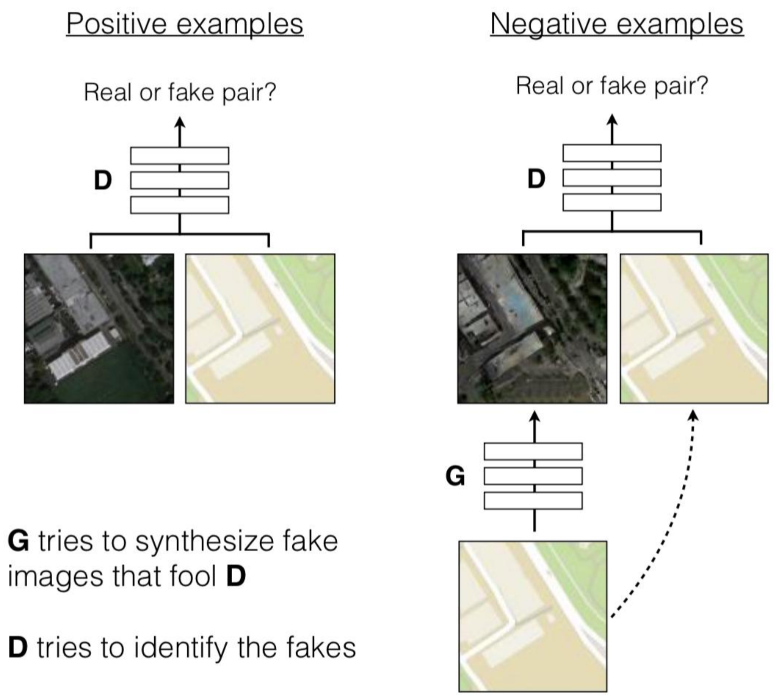
Image-to-image translation is the controlled conversion of a given source image to a target image. Examples include translating a photograph of a landscape from day to night or translating a segmented image to a photograph, synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images, among other tasks.

conditional adversarial networks are general-purpose solutions to image-to-image translation problems. These networks not only learn the mapping from input image to output image, but also learn a loss function to train this mapping. This makes it possible to apply the same generic approach to problems that traditionally would require very different loss formulations.

Just as GANs learn a generative model of data, conditional GANs (cGANs) learn a conditional generative model. This makes cGANs suitable for image-to-image translation tasks, where we condition on an input image and generate a corresponding output image.

Architecture

The GAN architecture consists of a generator model for outputting new plausible synthetic images, and a discriminator model that classifies images as real (from the dataset) or fake (generated). The discriminator model is updated directly, whereas the generator model is updated via the discriminator model. As such, the two models are trained simultaneously in an adversarial process where the generator seeks to better fool the discriminator and the discriminator seeks to better identify the counterfeit images.


Positive examples
Negative examples
Real or fake pair?
Real or fake pair?
D
D
G
G tries to synthesize fake images that fool D
D tries to identify the fakes

For our generator we use a “U-Net”-based architecture and for our discriminator we use a convolutional “PatchGAN” classifier, which only penalizes structure at the scale of image patches. Both generator and discriminator use modules of the form convolution-BatchNorm-ReLu

Objective function

GANs are generative models that learn a mapping from random noise vector z to output image y, $G : z \rightarrow y$

In contrast, conditional GANs learn a mapping from observed image x and random noise vector z, to y, G : {x, z} ➔ y

$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) =&\mathbb{E}_{x,y}[\log D(x, y)]+\\ &\mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]\end{aligned}$$

where G tries to minimize this objective against an adversarial D that tries to maximize it, i.e. G∗ = arg minG maxD LcGAN (G, D).

It is beneficial to mix the GAN objective with a more traditional loss, such as L2 distance. We'll using L1 distance rather than L2 as L1 encourages less blurring:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]$$

Our final objective is:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda\mathcal{L}_{L1}(G).$$

Business Problem

Given a satellite image, we have to generate an corresponding aerial map image.

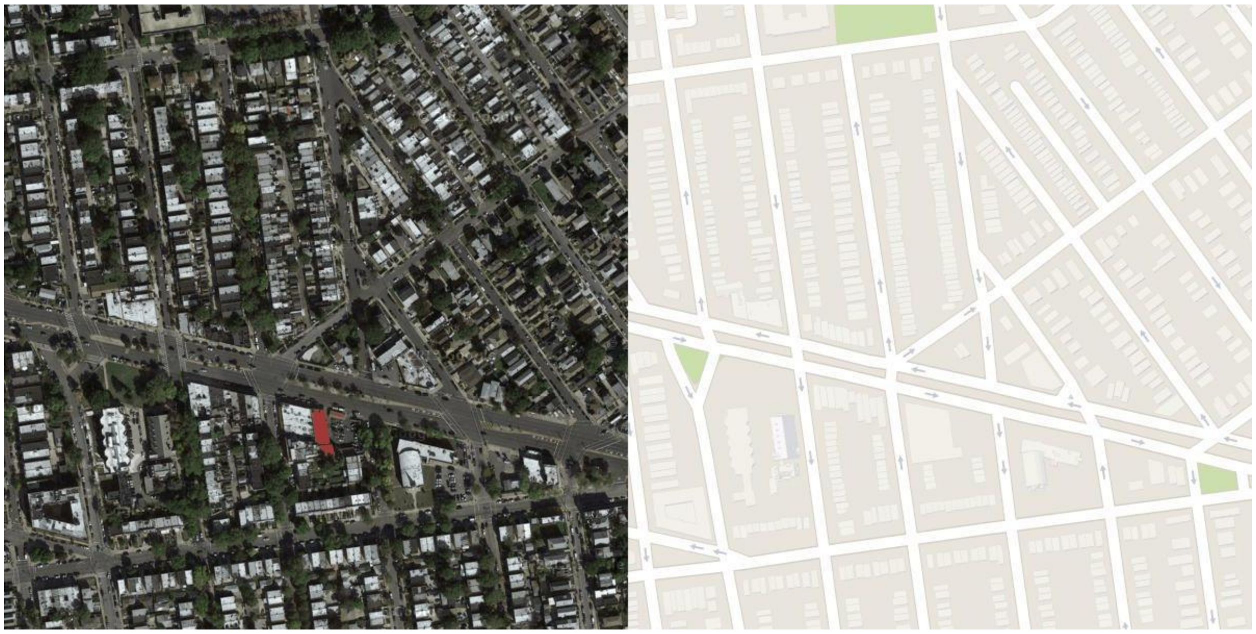
Business constraints

- The solution should not take hours to run. It should be time-efficient.

Dataset

Data Overview :

- Source: http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/
- There are 2 dataset files in maps.tar.tz i.e train and val.
- Train folder contains 1096 images while val contain 1098 items.
- Each image contains a satellite image and its corresponding map image.



Research-Papers/Solutions/Architectures/Kernels

1. Generative Adversarial Networks
    a. GAN consists of 2 parts:
        i. A Generative Model G that captures the data distribution
        ii. A discriminative Model D that estimates the probability that a sample came from training data rather than G
    b. The generative model generates samples by passing random noise through a MLP and the discriminative model is also a MLP.

c. Loss function:

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

We train D to maximize the probability of assigning the correct label (real or fake) to both training examples and samples from G.
We train G to minimize log(1 - D(G(z)))

d. Training approach: Alternate between k steps of optimizing D and one step of optimizing G.

e. Above mentioned loss function may not provide sufficient gradient for G to learn well. In the early stage of training, when G is poor, D can reject samples with high confidence because they are clearly different from the training data making log(1 - D(G(z))) saturate. We train G to maximize log(D(G(z))) instead of minimizing log(1 - D(G(z))).

2. Understanding GAN Loss Functions

a. The standard GAN loss function also know as min-max loss

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

b. The generator tries to minimize the above function while the discriminator tries to maximize it. However, in practice, it saturates for the generator, meaning that the generator stops training after a while.

c. Standard GAN loss can be broken into:
   i. Discriminator loss
   ii. Generator loss

d. Discriminator Loss - It penalizes itself for misclassifying real as fake, or a fake instance as real by maximizing the following:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

- log(D(x)) - probability that discriminator is rightly classifying the real image
- maximizing log(1-D(G(z))) would help it to correctly label the fake image that comes from the generator.

e. Generator loss- The generator loss is calculated from the discriminator's classification – it gets rewarded if it successfully fools the discriminator, and gets penalized otherwise. The following equation is minimized to training the generator:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

f. Non Saturating GAN loss:
   A variation of the standard loss function is used where the generator maximizes the log of the discriminator probabilities **log(D(G(z)))**.
   The generator seeks to maximize the probability of images being real, instead of minimizing the probability of an image being fake.

g. Challenges with GAN loss:
   i. **Mode Collapse:**
      We would want our GAN to produce a range of outputs. Instead, through subsequent training, the network learns to model a particular distribution of data, which gives us a monotonous output
   ii. **Vanishing Gradient:**
      This phenomenon happens when the discriminator performs significantly better than the generator.

h. Alternate GAN loss functions:
   i. **WGAN** (Wasserstein Generative Adversarial Network):
      1. It tackles the problem of Mode Collapse and Vanishing Gradient.
      2. The activation of the output layer of the discriminator is changed from sigmoid to a linear one.
   ii. **CGAN** (Conditional Generative Adversarial Network):
      Both the generator and the discriminator are fed with some extra information y which works as an auxiliary information

3. Implementing A GAN in Keras
    a. Author is implementing FC- GAN as well as DC-GAN on the MNIST dataset.
    b. For FC-GAN, Dense layers with LeakyRelu Activation are used.
    c. A sequential structure of 256 - LRelu - 512 - LRelu - 1024 - LRelu - Output(tanh Activation) is used for Generator.
    d. As images are normalized in range of [-1,1], tanh activation is being employed in the output layer of the generator.
    e. For discriminator, a sequential model of 1024-LRelu-512-LRelu-256-LRelu-Output(sigmoid) is used.
    f. After initializing both generator and discriminator, discriminator trainable is set to false as the generator is not going to be trained directly, both will be combined into a single model and trained. This allows the generator to understand the discriminator so it can update itself more effectively.
    g. Label smoothing is used which helps the discriminator in reducing sparse gradients.
    h. Similar process is used while training DCGAN. Instead of using FC layer, DCGAN uses Conv layer

4. Image-to-Image Translation with Conditional Adversarial Networks
    a. U-Net based architecture is used for the generator and a convolutional PatchGAN classifier is used for the discriminator.
    b. Conditional GAN learns mapping from observed image x and random noise vector z, to y, G: {x,z} -> y. Whereas, GAN learns mapping from random noise vector to output image y, G z -> y.
    c. Objective function:

$$\begin{aligned}\mathcal{L}_{cGAN}(G,D) =&\mathbb{E}_{x,y}[\log D(x,y)]+\\ &\mathbb{E}_{x,z}[\log(1-D(x,G(x,z))],\end{aligned} \quad (1)$$

       G tries to minimize the objective whereas D tries to maximize it.
    d. Final objective:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda\mathcal{L}_{L1}(G).$$

The generator is tasked to not only fool the discriminator but also to be near the ground truth output in an L2 sense. Using L1 distance rather than L2 as L1 encourages less blurring.

e. Both generator and discriminator use modules of the form convolution-BatchNorm-ReLu.
f. Generator:
   U-Net decoder: CD512-CD1024-CD1024-C1024-C1024-C512 -C256-C128
g. 286 × 286 discriminator: C64-C128-C256-C512-C512-C512
h. Ck denotes a Convolution-BatchNorm-ReLU layer with k filters. CDk denotes a Convolution-BatchNormDropout-ReLU layer with a dropout rate of 50%.
i. All convolutions are 4× 4 spatial filters applied with stride 2.

5. How to Develop a Pix2Pix GAN for Image-to-Image Translation

a. Dataset used: Map Dataset containing 1097 train images and 1099 val images
b. Image contains satellite image on left and google maps image on right.
c. All images are resized from original size of 1200x600 to 512x256. And then splitted into 2 i.e satellite image and map image.
d. Model is implemented as per the original research paper.
e. All conv are of size 4x4 with stride of 2. LeakyRelu is used for activation
f. For Discriminator, architecture of C64-C128-C256-C512-C512-C1(sigmoid) is used.
g. For generator, U-Net architecture of C64-C128-C256-C512-C512-C512-C512-C512(bottleneck)-CD512-CD512-CD512-C512-C256-C128-C64-C3(tanh) is used.
h. All images are normalized in range of [-1,1]
i. Model has been trained for 100 epochs and model is saved every 10 epochs and samples are printed using saved model every 10 epochs

First Cut Approach

1. For the dataset, I want to model on the SatToMap Image dataset (Maps dataset).
2. For image preprocessing, first resizing it to 256x256 and normalizing it to [-1,1].
3. For model architecture, implement the same arch. as mentioned in the last blog (How to Develop a Pix2Pix GAN for Image-to-Image Translation).
4. Use of label smoothing for true class instead of hard labels.
5. We can use image augmentation to generate more train samples but i feel that given number of samples are enough.

Notes when you build your final notebook:

1. You should not train any model either it can be a ML model or DL model or Countvectorizer or even simple StandardScalar
2. You should not read train data files
3. The function1 takes only one argument "X" (a single data points i.e 1*d feature) and the inside the function you will preprocess data point similar to the process you did while you featurize your train data
    a. Ex: consider you are doing taxi demand prediction case study (problem definition: given a time and location predict the number of pickups that can happen)
    b. so in your final notebook, you need to pass only those two values
    c. def final(X):

preprocess data i.e data cleaning, filling missing values etc

compute features based on this X

use pre trained model

return predicted outputs

final([time, location])

    d. in the instructions, we have mentioned two functions one with original values and one without it

    e. final([time, location])   # in this function you need to return the predictions, no need to compute the metric

    f. final(set of [time, location] values, corresponding Y values) # when you pass the Y values, we can compute the error metric(Y, y_predict)

4. After you have preprocessed the data point you will featurize it, with the help of trained vectorizers or methods you have followed for your train data

5. Assume this function is like you are productionizing the best model you have built, you need to measure the time for predicting and report the time. Make sure you keep the time as low as possible

6. Check this live session: https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/4148/hands-on-live-session-deploy-an-ml-model-using-apis-on-aws/5/module-5-feature-engineering-productionization-and-deployment-of-ml-models