Report for Pix2Pix project

Introduction

My project is a picture-to-picture (Pix2Pix) model. Pix2pix models are a type of deep learning model used for image-to-image translation. This means they learn how to change one image into another related image. For example, a pix2pix model can learn how to turn black-and-white photos into colour images, maps into satellite photos, or sketches into realistic pictures. The model is trained on pairs of images, where each input image has a correct target image. By seeing many such pairs, the model learns the relationship between the input and the output. Pix2pix models are useful in many practical tasks such as photo enhancement, medical imaging, and computer vision research.

My dataset is from I*mage-to-Image Translation with Conditional Adversarial Networks* (2017) paper. The original paper was focusing on many different pictures to pictures transitions. For my model I only used part of dataset which consist of pairs of satellite photos and street map. In this report I will elaborate on preprocessing data, architecture of my model and problems I have encountered while working on this project.

Preprocessing

My dataset is 2194 pairs of images. I downloaded two folders from Kaggle (https://www.kaggle.com/datasets/alincijov/pix2pix-maps). Function *gener_file_names* creates a list of strings which are directories of all files that will be the source for my model. Following function *split_t_v_t* shuffles and splits all files in 3 subsets: train – 70%, validation – 15%, and test – 15%. Next function in the code is *process_file*. It processes files with cv2 library, the function resizes images to dimensions which are powers of 2. Then image is being processed to floating-tensor point, afterwards by dividing by 255 pixels range become [0, -1] instead of [0, 255] and the last step is transforming range to [-1, 1] which is preferable for tanh part of neural network architecture. Function *inp_outp_tuple* is responsible for applying *process_file* function for each file in the dataset.

Function *cv2_to_tensor* is responsible for transforming data to torch tensor and then permuting dimension in following order colour, height, width. This help function will be used in *pix2pixDS* class. *Pix2pixDS* is a subset of Dataset class from torch.utils.data library, it contains all required functions: __*init*__, __*len*__ and __*getitem*__. The __*getitem*__ returns x and y processed with already described *cv2_to_tensor* function. With ready *Dataset train*, *validation* and *test* objects the code is generating three *DataLoaders* accordingly.

U-net architecture

The most important part of Pix2Pix model is well designed architecture. As in Berkley (2017) paper I designed my architecture as generative adversarial network (GAD). The characteristic of GAD is that it contains 2 models. First Model is called generator, and the other is called discriminator. Generator is a classic encoder-decoder model like U-net, which given an input and golden label, tries to learn how to generate a street map based on satellite photo. After each time generator will generate a fake image the discriminator model acts. Discriminator is being fed with two pairs of images (input and golden label) and (input and image generated by generator). The role of discriminator is to judge if the second image in the pair is real of fake. The relationship between these two models can be described as zero-sum game. At first discriminator will be much more successful with the differentiating between real and fake images. As the training goes on generator will be getting better at generating street maps pictures, and it will be harder for discriminator to correctly differentiate between pairs.

At first my generator architecture was very similar to U-net architectures one uses in classificational tasks like original biomedical U-net. Hence it wasn't very successful the following U-net version was adjusted to GAN needs. Classic U-net focuses on segmentation, and GAN adjusted U-net needs to conditional adversarial learning. In GAN model it is preferable to use one convolution per block. The flow of information is more controlled, and it help to avoid over-parametrisation. The realism in GAN architecture is forced rather by feedback from discriminator architecture. Too expressive generated image would undermine the roles of two GAN's models. Another difference is that GAN's generator shouldn't use pooling, as opposed to that it's preferable it will use strided convolution. The down sampling convolution is learnable, as opposed to pooling which harms image synthesis, and deletes phase information (where structures are located in the image). Another change that was required was to allow leaky ReLU in encoder part of U-net. Leaky ReLUs are preventing death of neurons by allowing very small value. Thanks to that weights can be updated during weight propagation. ReLU in decoder part stay *not leaky* to help generate sharper structures. Because advised batch number size for pix2pix models is 1, it's better to used *InstanceNorm* type of normalisation as opposed to *BatchNorm* normalisation which would be used models with bigger batches. It allows independent normalisation on an image level.

The whole U-net model is made of 3 *nn.Module* classes. *DownBlock* that groups steps needed for downsampling. It uses only one convolutional layer, leaky ReLU and stride. *UpBlock class* is responsible for upsampling steps in U-net architecture, with help of transposed convolution, not leaky ReLUs and instance normalisation. The *UNetGenerator* uses first classes to perform U-net training. Thank to very small batches, memory allows using multi 7 steps sampling. Dropout is being used in the first stages of the decoder, to encourage diversity. In the last final stage Tanh function is being applied to make outcome suitable for adversarial training.

Discriminator architecture

Discriminator architecture is much simpler compared to generator. It all is coded only in one class called *Discriminator*. As mentioned earlier discriminator role is to judge if pair of satellite picture image, and street map image contains real or fake street map. In forward function two objects of the pair are being concatenated with *torch.cat* function. As a result, discriminator works with 6 channels image. With that input discriminator evaluates whether the generated image is realistic. This is a PatchGan model, so discriminator focuses on realism of single patches rather than entire image. Thanks to that that model can be smaller and focus on prise sharper images. The classic sequence of convolutional layers with instance normalization and leaky ReLU activations reduces spatial resolution and increases feature depth. The final convolutional layer outputs two-dimensional map of realism.

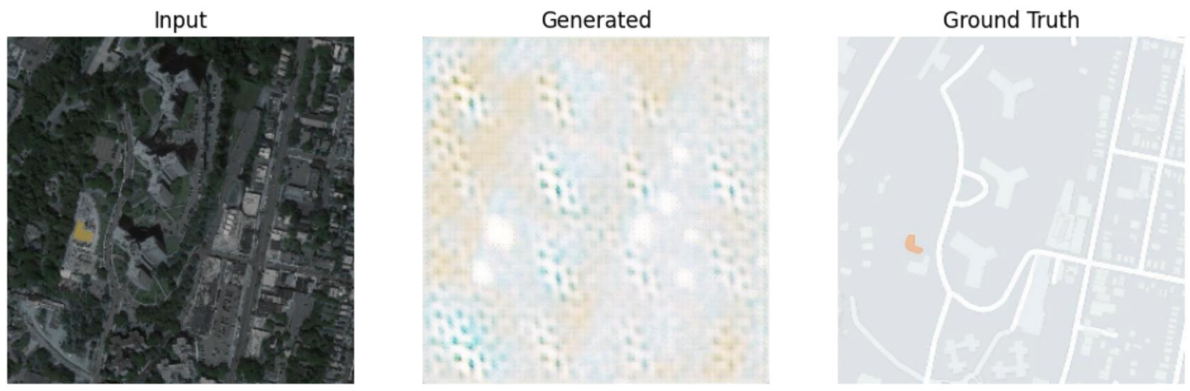
Training

Training part uses two models: generator and discriminator. Generator is being trained with the combination of two loss functions adversarial loss and L1 reconstruction loss. Adversarial component focuses on photorealistic outputs and the L1 function try to keep similarity close to the ground truth. In both cases the optimiser that is been used is Adam. Both optimisers have tuned momentum parameters to stabilise adversarial learning. First the generator is generating the fake street map image. Then the discriminator is evaluating both pairs each. With that loss function for discriminator is being returned. And only after that the loss function for Generator is being fed back. The model is being evaluated with validation set after each epoch. Every 10 epochs the state dictionary for the epoch is being saved. This will be used in evaluation code.

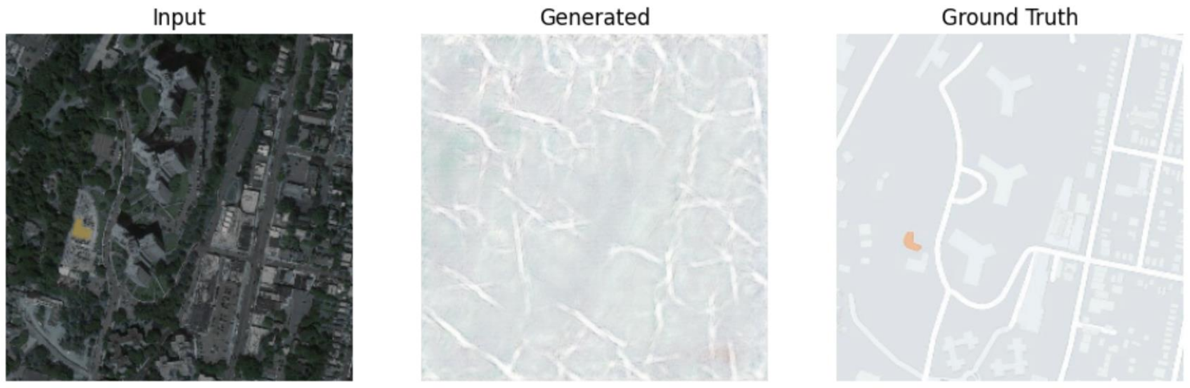
Evaluation

Since this model is based on image generation, best method to evaluate is human gaze. Below I present few comparisons of same satellite image transformation on different stages of model training. The images are from test set.

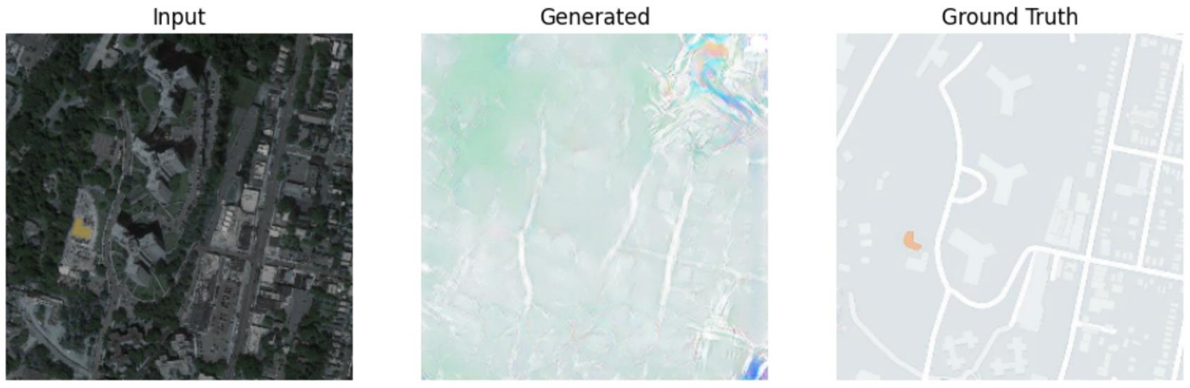
Epoch 0



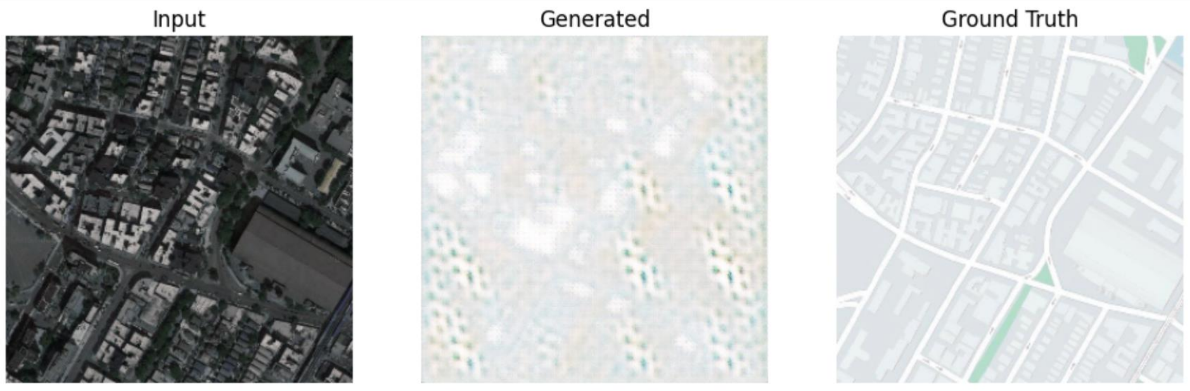
Epoch 10



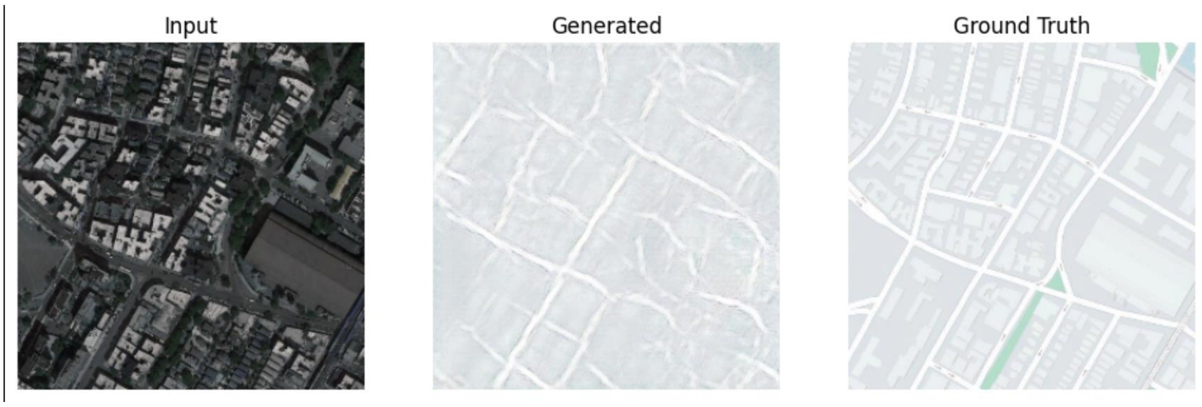
Epoch 170



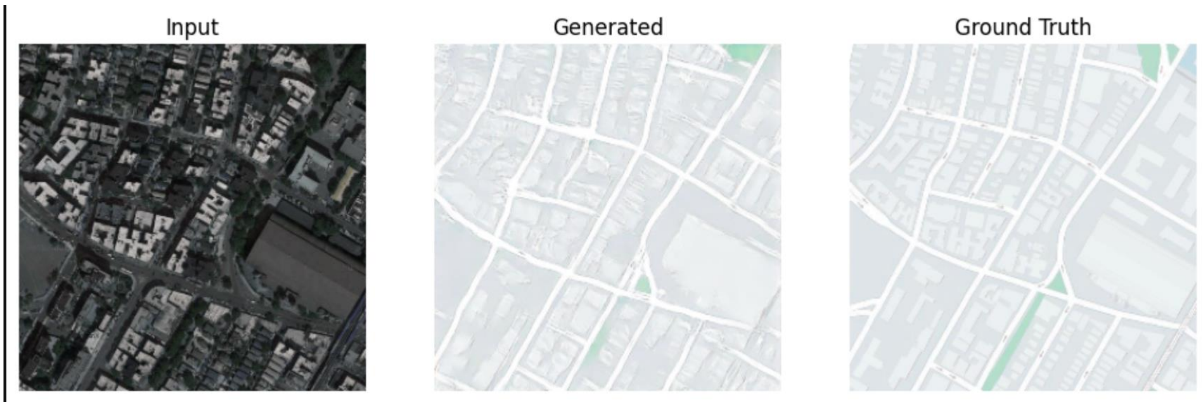
Epoch 0



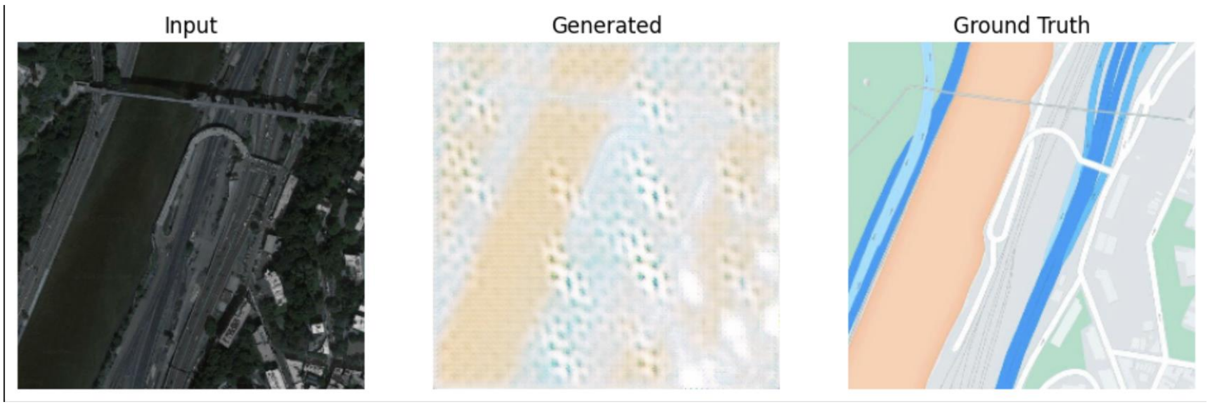
Epoch 10



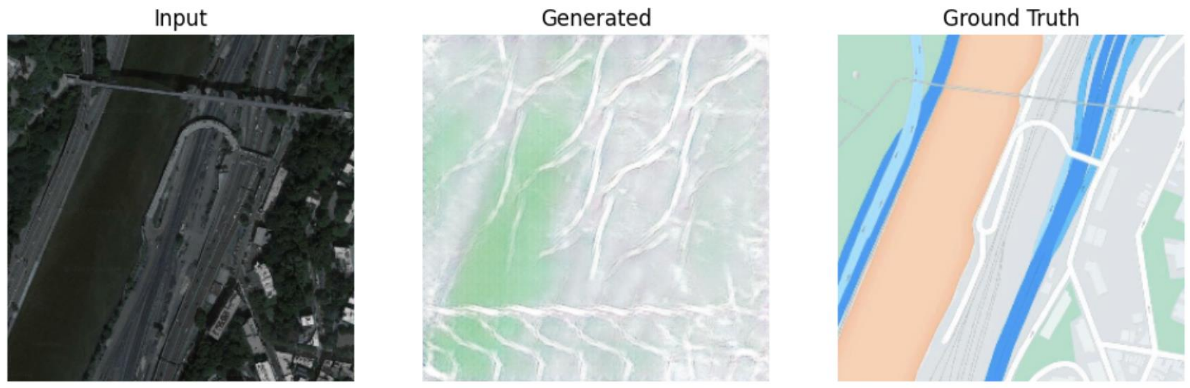
Epoch 170



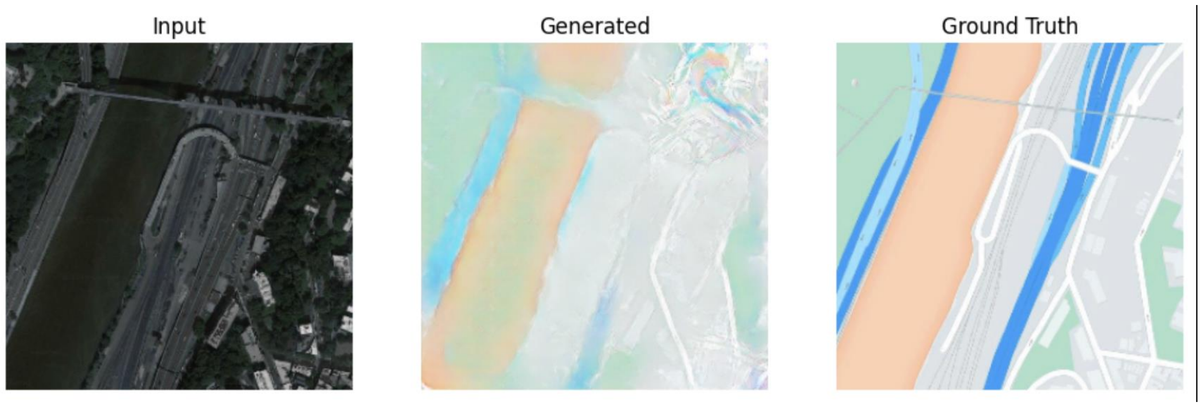
Epoch 0



Epoch 20



Epoch 170



Literature:

P. Isola, J. -Y. Zhu, T. Zhou and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 5967-5976,

Stanislaw Lorys