University of Michigan
EECS 442: Computer Vision
Fall 2020. Instructor: Andrew Owens.

Problem Set 6: Image Translation

Posted: Wednesday, October 14, 2020	**Due:** Wednesday, October 21, 2020

Please follow our instruction to convert the your Colab notebook to a PDF file and submit the PDF file to Gradescope. For your convenience, we have included the PDF conversion script at the end of the notebook. If you have any issue converting the notebook file, please refer to the Piazza post @432 for more detailed guidance. Nothing needs to be submitted to Canvas.

The starter code can be found at:
https://colab.research.google.com/drive/16NRU7us4MM6D4kawO814G5Njv3deWO3A?usp=sharing

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 6.1 *Implement pix2pix (**9 points**)*

At the last EECS 442 shareholders' meeting, concerns were raised about the class's continued lack of profitability. Investors are now pressuring the class to pursue new markets, and are encouraging the class to pivot into the lucrative fashion sector. In light of these developments, the highly theoretical "GAN convergence bounds" problem set that we had originally planned for this week will, sadly, be shelved. Instead, we will implement a system that allows our new customers to generate pictures of shoes from only a sketch.

In this problem set, we will implement an image-to-image translation method, based on pix2pix [1]. We will train the pix2pix model on the *edges2shoes* dataset [1, 2] to translate images containing only the edges of a shoe, to a full image of a shoe. The edges are automatically extracted from the real shoe images. Some example edge/image pairs are shown in Figure 1.

The pix2pix model is based on a conditional GAN (Figure 2). The generator G maps the source image x to a synthesized target image. The discriminator takes both the source image and generated target image as its inputs, and outputs `True` or `False` labels based on the quality of the synthesized image.

1. We will first build data loaders for training and testing. For the training process, we will use a batch size of 4. During testing, we will process 5 images in a single batch, so that we can visualize several results at once. (**1 point**)

Figure 1: Example sketch-image pairs from the edges2shoes dataset. The edges are extracted with HED edge detector [3] from the raw shoe images. A model trained on this dataset can also work with user-provided sketches.

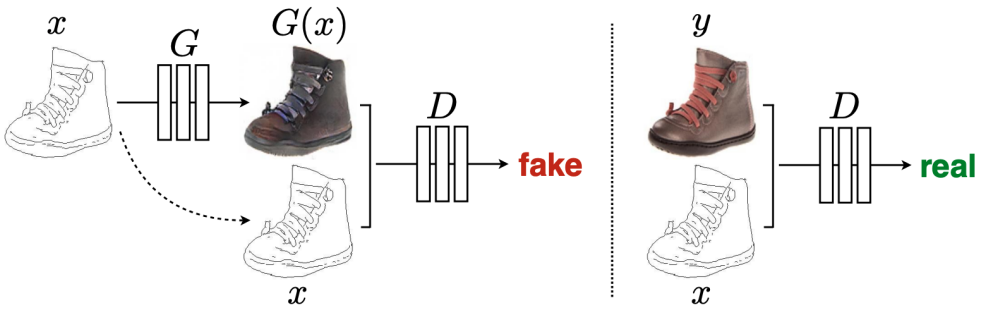

Figure 2: Conditional GAN for image translation.Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real edge, photo tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

2. In the next step, we will define the network based on the architectures from the paper. The generator follows a U-net architecture, where the activations from the encoder are concatenated with the inputs to the decoder (Figure 3). For more details, please see the Colab notebook.

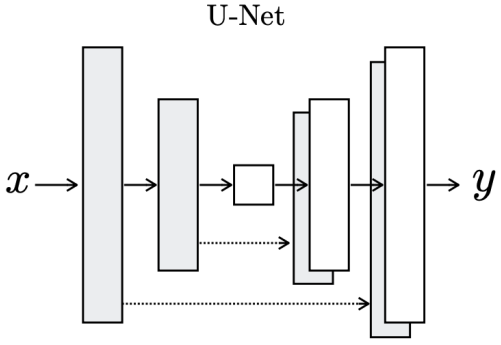

Figure 3: U-net architecture.

As a reminder: let Ck denote a Convolution-BatchNorm-ReLU layer with k filters. All

convolutions are 4×4 spatial filters applied with stride 2. Convolutions in the encoder, and in the discriminator, downsample the input by a factor of 2, whereas in the decoder they upsample the input by a factor of 2.

(a) Generator architectures (**2.5 points**)

The U-Net encoder-decoder architecture consists of:
U-Net encoder:
C64-C128-C256-C512-C512-C512-C512-C512
U-Net decoder:
C512-C512-C512-C512-C256-C128-C64-C3

After the last layer in the decoder, a convolution is applied to map to the number of output channels, which is 3 in our problem, followed by a tanh function. As a special case, batch normalization is not applied to the first C64 layer in the encoder. All ReLUs in the encoder are leaky, with slope 0.2, while ReLUs in the decoder are not leaky.

(b) Discriminator architectures (**2.5 points**)

The discriminator architecture is:
C64-C128-C256-C512

After the last layer, a convolution is applied to map to a 1-dimensional output, followed by a sigmoid function. As an exception to the above notation, batch normalization is not applied to the first C64 layer. All ReLUs are leaky, with slope 0.2.

Hint: Using torch.nn.functional.leaky_relu for leaky ReLU.

3. For optimization, we'll use the Adam optimizer. Adam is an optimization algorithm that improves upon SGD with momentum by adaptive tuning the learning rate of each model parameter throughout the training process. For our model training, we will use a learning rate of 0.0002, and momentum parameters $\beta_1 = 0.5, \beta_2 = 0.999$. If you want to learn more about Adam algorithm, you can refer to the Deep Learning book [4].(**0 point**)

4. Now, we will implement the training routine and start trainig the models. (**3 points**)
The conditional GAN (cGAN) loss function can be written as:

$$\mathcal{L}_{cGAN}(G, D) = \frac{1}{N}\sum_{i=1}^{N} \log D(x_i, y_i) + \frac{1}{N}\sum_{i=1}^{N} \log(1 - D(x_i, G(x_i)). \tag{1}$$

We also add an L1 loss to the total total loss function:

$$\mathcal{L}_{L1}(G) = \frac{1}{N}\sum_{i=1}^{N} [\|y_i - G(x_i)\|_1] \tag{2}$$

The L1 loss is good at capturing the overall structure of the source image while the cGAN loss is good at learning the details. In other words, L1 loss is for learning the low

Figure 4: Left: a sketch drawn by a GSI. Right: the synthesized shoe image.

frequency components and cGAN loss is for learning the high frequency components. In each epoch, we first train discriminator D by using the average loss of real image and fake image. Then train generator G by using the following loss:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G). \tag{3}$$

In this homework, you will train two different models: one with only L1 loss, the other with the equation above and $\lambda = 100$. Train the network for at least 20 epochs (10 epochs for the model with only L1 loss,) but you are welcome to train longer to obtain better results.

In the specified text cell of the Colab notebook, please comment on the difference between the translated images obtained from L1 only and L1 + cGAN.

Also, please plot the G/D loss history vs. iteration of two models in separate plot.

Note: training each epoch will take less than 2 minutes. If your training takes significantly longer than this, there is likely a mistake in your implementation.

5. (**Optional**) After the pix2pix model has been trained on this dataset, we can apply the trained generative model to translate any user-provided shoe sketch to a synthetic image. A sketch drawn by a GSI and the synthesized image are plotted in Figure 4.

   Please draw a shoe in the sketch panel we provide in the Colab notebook and translate the sketch to a shoe image with the trained model. (**0 point**)

Problem 6.2 *Understanding pix2pix (1 point)*

The network architecture that we used in the previous section is often called a *PatchGAN*. This is because the discriminator classifies individual image patches, rather than assigning a score to a whole image. To help you better understand the concept of classifying on a patch level, we calculate the patch size. This is equivalent to the *receptive field size* of the final convolutional layer of the network (Figure 5)).

Given the inputs with the size $256 \times 256 \times 6$ for the discriminator, if we use the discriminator network architectures from 2(b), write down the size of each neuron's receptive field after each layer. Please put your answer in the notebook section 6.2. There is **no** need to submit a separate file to Canvas. (**1 points**)

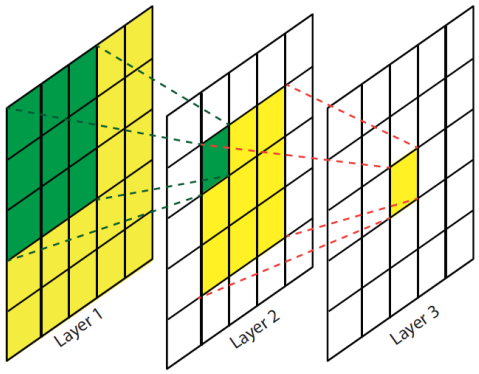

Figure 5: Visualization of receptive fields. As we move to higher layers, the receptive field of a single neuron activation is increasing rapidly [5].

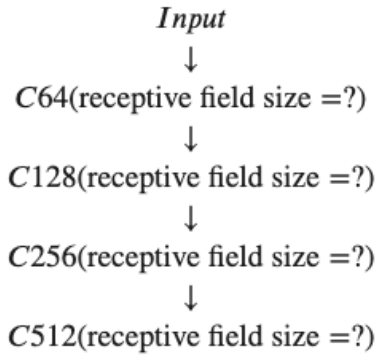

Figure 6: Discriminator architecture. Please fill in the receptive field size in the Colab notebook.

Please note that receptive field size is *not* the same as the width/height of a given layer. If you need a review, please refer to the lecture slides on convolutional networks, as well as to this very readable paper [6].

References

[1] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[2] A. Yu and K. Grauman. Fine-grained visual comparisons with local learning. In *Computer Vision and Pattern Recognition (CVPR)*, Jun 2014.

[3] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of IEEE International Conference on Computer Vision*, 2015.

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[5] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. *Remote sensing*, 9(5):480, 2017.

[6] André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11):e21, 2019.