University of Michigan
EECS 442: Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 6: Image Translation

Posted: Wednesday, October 13, 2021 **Due:** Wednesday, October 27, 2021

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope. We have included the PDF conversion script at the end of the notebook. Nothing needs to be submitted to Canvas.

The starter code can be found at:
https://colab.research.google.com/drive/1ZIbFTNH8CxnrZItq8fwoCx6arglKk-IB?usp=sharing

Problem 6.1 *(12 points) Implement pix2pix*

In this problem set, we will implement an image-to-image translation program 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.

Optionally, you can also train the pix2pix model on the *Pokémon Images Dataset* [3, 4]. Some example edge/image pairs are shown in Figures 1 and 2.

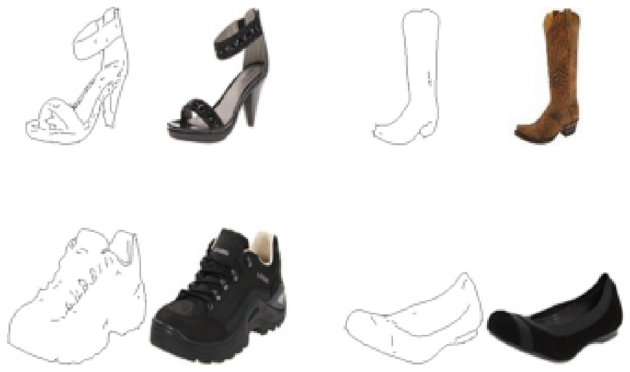

Figure 1: Example sketch-image pairs from the edges2shoes dataset. The edges are extracted with HED edge detector [5] from the raw shoe images. A model trained on this dataset can also work with user-provided sketches.

The pix2pix model is based on a conditional GAN (Figure 3). The generator G maps the source image x to a synthesized target image. The discriminator takes both the source image and predicted target image as its inputs, and predicts whether the input is real or fake.

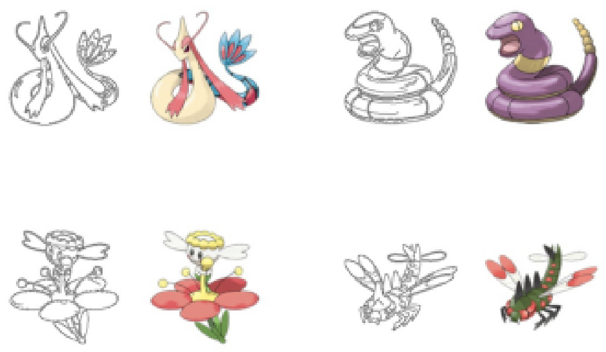

Figure 2: Example sketch-image pairs from the Pokemon Images Dataset dataset. The edges are extracted with a Canny edge detector from the Pokemon images [4].

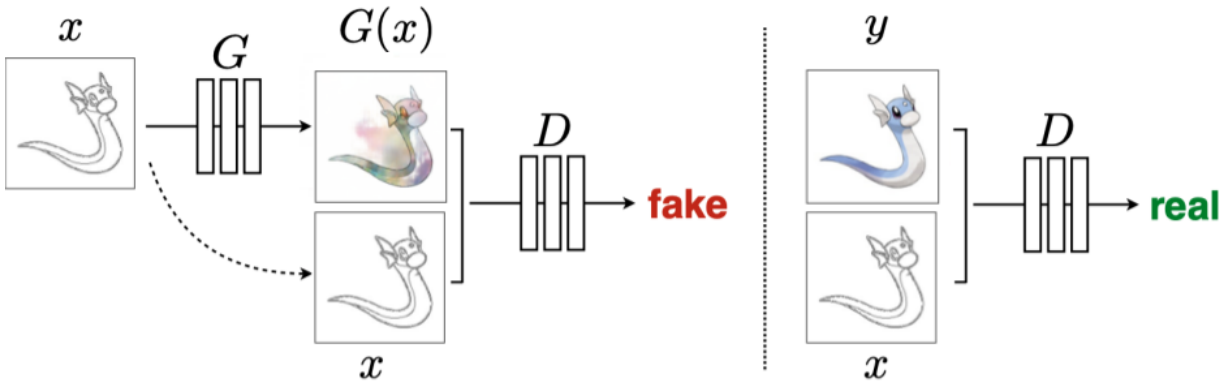

Figure 3: Conditional GAN for image translation.Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real edge, photo tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

1. (**1 point**) 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. Please complete the Edges2Image class and fill in the TODOs in that cell.

   Hint: please use the `DataLoader` from `torch.utils.data`

2. (**6 points**) Next, we will define the network based on the architecture 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 4). We have included a toy U-net example in the Colab notebook.

   As a reminder: let Ck denote a Convolution-BatchNorm-ReLU layer with k filters. All convolutions are 4×4 spatial filters applied with padding 1, and stride 2, except for the last 2 layers in the discriminator, which have stride 1. Convolutions in the encoder and the discriminator downsample the input by a factor of 2, while convolutions in the decoder upsample the input by a factor of 2.

   (a) (**3 points**) Generator architectures

   The U-Net encoder-decoder architecture consists of:
   U-Net encoder:

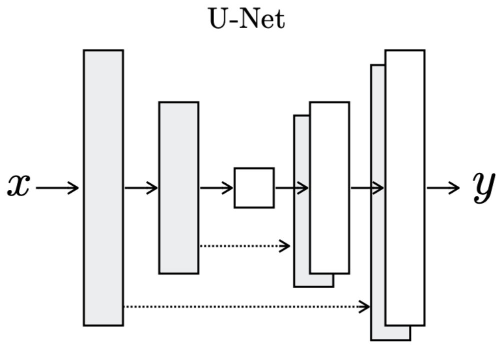

Figure 4: U-net architecture.

C64-C128-C256-C512-C512-C512-C512-C512
U-Net decoder:
C512-C512-C512-C512-C256-C128-C64-C3

As a special case, batch normalization is not applied to the first C64 layer in the encoder. All nonlinearities in the encoder are Leaky ReLUs, with slope 0.2, while the nonlinearities in the decoder are ReLUs. 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.

Please complete the generator class in the starter code.

Hint: you can use `torch.cat` to concatenate the decoder and the encoder inputs

(b) (**3 points**) Discriminator architectures

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

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

Please complete the discriminator class in the starter code.

Hint: Using `torch.nn.functional.leaky_relu` for Leaky ReLU.

3. (**1 point**) For optimization, we'll use the Adam optimizer. Adam is similar to SGD with momentum, but it also contains an adaptive learning rate for each model parameter. If you want to learn more about Adam algorithm, please refer to the Deep Learning book [6]. For our model training, we will use a learning rate of 0.0002, and momentum parameters $\beta_1 = 0.5, \beta_2 = 0.999$. Please set up `G_optimizer` and `D_optimizer` in the train function.

4. (**4 points**) Now, we will implement the training routine and start training the models.

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:

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

Each iteration, we first train discriminator D by using the average loss of real image and fake images. We then train generator G by using the following loss:

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

You will train two different models: one with only L1 loss, the other with Equation 3 and $\lambda = 100$. Train the network for at least 20 epochs (10 epochs is enough for the model with only L1 loss). You are welcome to train longer, though, to potentially obtain better results.

Please complete the following tasks:

- In the specified text cell of the Colab notebook, comment on the difference between the translated images obtained from L1 only and L1 + cGAN.
- Show the history of the generator's BCE and L1 losses and the discriminator's loss vs. iteration of the $c = 100$ model in 3 separate plots.
- Show the history of the generator and the discriminator L1 loss vs. iteration of the L1 only model in 2 separate plots.
- In the specified text cell of the Colab notebook, comment on the difference among the history of loss plots for the L1 only and L1 + cGAN models. Specifically, what are the behaviors of BCE and L1 losses for the $c = 100$ model?

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 0 points**) After the pix2pix model has been trained on this dataset, we can apply the trained generative model to translate any user-provided sketch to a synthetic image. A Pokemon sketch drawn by an IA and the synthesized Pokemon image are plotted in Figure 6.

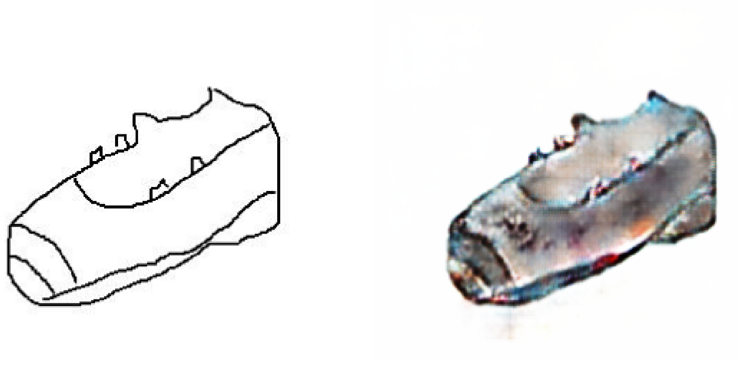

Figure 5: Left: a sketch drawn by a GSI. Right: the synthesized shoe image.

Please draw a shoe in the sketch panel we provide in the Colab notebook and translate it to a shoe image with the trained model. Feel free to post your generated image to a Piazza thread.

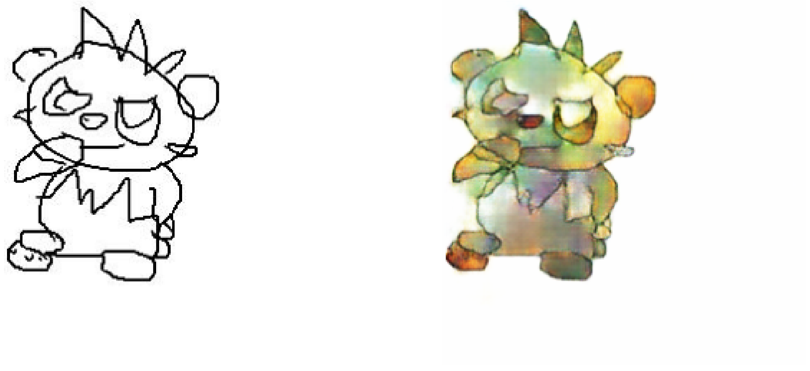

Figure 6: Left: a sketch drawn by an IA. Right: the synthesized Pokemon image.

6. (**Optional 0 points**) Your EECS 442 IA aspires to become a Pokemon trainer after graduating from college, but the Pokemon battles are too fierce. His strategy is to create new Pokemon to surprise his opponents. With your Computer Vision skills, you will help him write a program to generate images of Pokemon from simple sketches. To complete this task, you will need to

   - Set Pokemon_trainer = True
   - Modify the data loader to load the `pokemon` folder instead of `mini-edges2shoes`
   - Rerun all previous cells to retrain the model (except for L1 only, which is unnecessary)
   - Sketch your Pokemon!

   Please also share your generated Pokemon in the Piazza thread with your classmates.

Problem 6.2 *(1 point) Understanding pix2pix*

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 7).

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**)

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 [8].

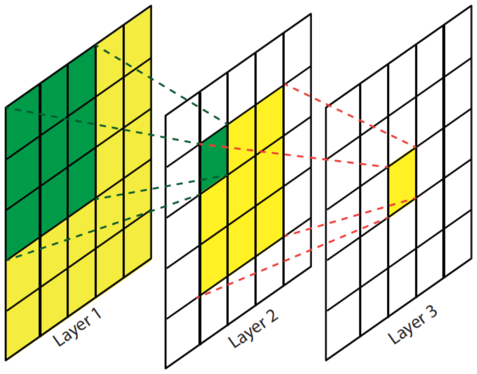

Figure 7: Visualization of receptive fields. As we move to higher layers, the receptive field of a single neuron activation is increasing rapidly [7].

$$\begin{tikzcd}
\textit{Input} \arrow[d] \\
C64(\text{receptive field size } =?) \arrow[d] \\
C128(\text{receptive field size } =?) \arrow[d] \\
C256(\text{receptive field size } =?) \arrow[d] \\
C512(\text{receptive field size } =?)
\end{tikzcd}$$

Figure 8: Discriminator architecture. Please fill in the receptive field size in the Colab notebook.

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] Pokemon images dataset. `https://www.kaggle.com/kvpratama/pokemon-images-dataset`.

[4] zaidalyafeai.github.io. `https://github.com/zaidalyafeai/zaidalyafeai.github.io/tree/master/pix2pix/datasets`, 2018.

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

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

[7] 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.

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