University of Michigan
EECS 442: Computer Vision
Fall 2022. Instructor: Andrew Owens.

Problem Set 6: Image Synthesis

Posted: Wednesday, October 12, 2022 **Due:** Wednesday, October 26, 2022
Submission instructions:

- Canvas:
  - Colab notebook (`.ipynb`) containing solutions and visualizations for 6.1, 6.2 and 6.3. Part of 6.1(c), 6.2 and 6.3(c) will be written questions. Please put your answers in the TODOs in the notebook and don't submit an additional file to GradeScope. Before submitting, please make sure to rename the file to `PS<this_ps_number>_<uniqname>.ipynb`.
- Gradescope
  - `.pdf` version of Colab notebook (conversion instructions here). If this instruction does not work, look into this additional method. For your convenience, we have included the PDF conversion script at the end of the notebook.

The starter code can be found at:
https://drive.google.com/file/d/12pxuMjssm8GYzvUG1hWBegLuDcs1ZG77/view?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.

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 predicted target image as its inputs, and predicts whether the input is real or fake.

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

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.

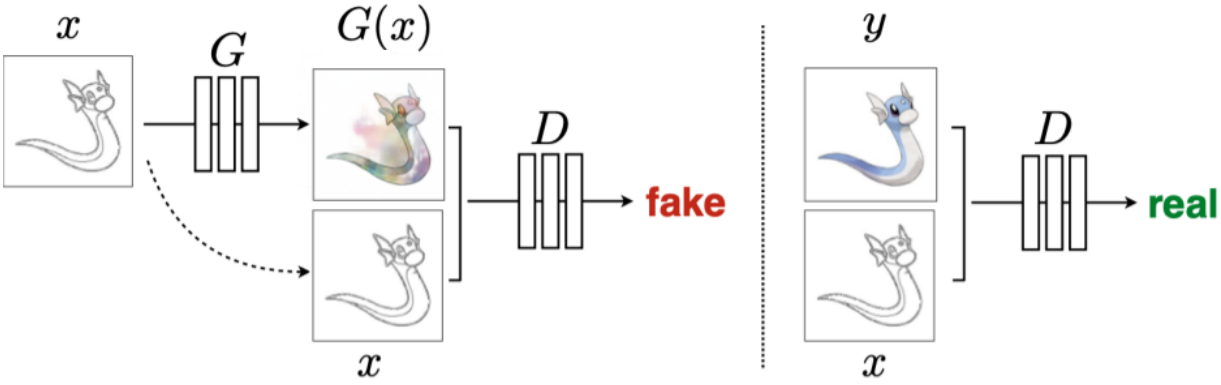


Figure 2: Conditional GAN for image translation.Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real edge, photo tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map. The *Pokémon* images shown here come from *Pokémon Images Dataset* [4, 5]. You can also train a GAN using this dataset and generate your own *Pokémon*, if you'd like! Here's a link to the dataset: https://github.com/zaidalyafeai/zaidalyafeai.github.io/tree/master/pix2pix/datasets

concatenated with the inputs to the decoder (Figure 3). We have included a toy U-net example in the Colab notebook.

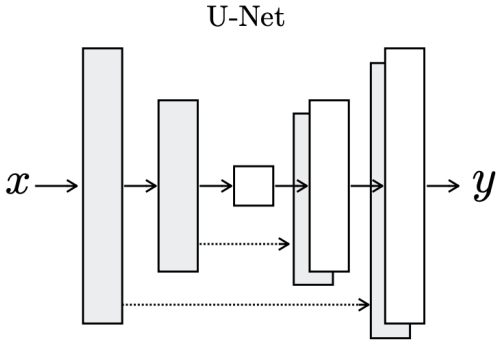


Figure 3: U-net architecture.

As a reminder: let Ck denote a Convolution-BatchNorm-ReLU layer with k filters. All convolutions are 4×4 spatial filters applied with 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:
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 and last layers 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 and last layers. 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, please refer to the deep learning book [6]. For our model training, we will use a learning rate of 0.0002, and momentum parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$. 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 shoe sketch drawn by a GSI and the synthesized shoe image are plotted in Figure 4.

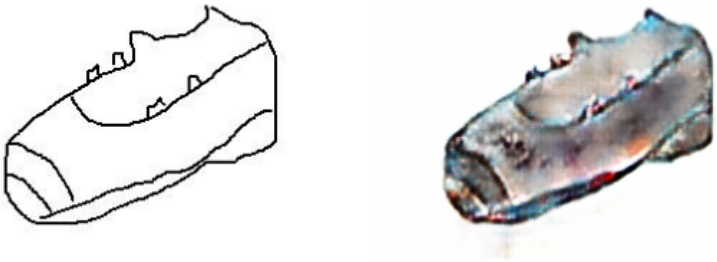


Figure 4: Left: a sketch drawn by a GSI. Right: the synthesized shoe image.

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.

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 5).

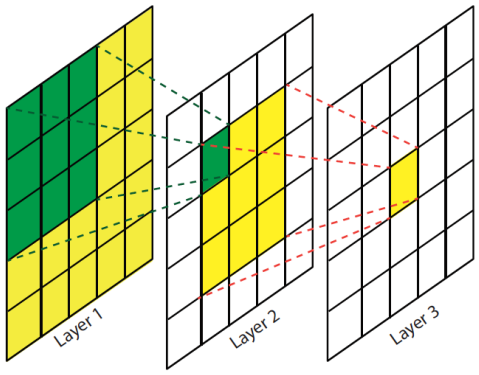


Figure 5: Visualization of receptive fields. As we move to higher layers, the receptive field of a single neuron activation is increasing rapidly [7].

Given $256 \times 256 \times 6$ images and 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. (**1 points**)

Input
↓
$C64$(receptive field size =?)
↓
$C128$(receptive field size =?)
↓
$C256$(receptive field size =?)
↓
$C512$(receptive field size =?)

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 [8].

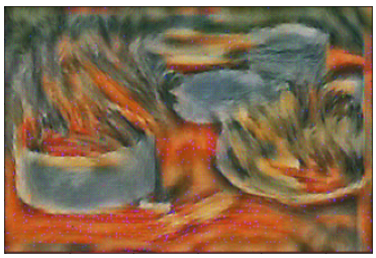
Problem 6.3 ***(4 points)*** *Style Transfer* (**EECS 504 Only**)

In this problem, we will implement the loss functions for the neural artistic style transfer method of Gatys et al. [9]. We take two images as input: one that defines the *content* and another that defines the *style* (Figure 7). We have provided five paintings that define the styles, and images COCO that define the content [10].

Two losses are needed to accomplish style transfer: the content loss and the style loss. Both

(a) Content image

(b) Style image



(c) Result

Figure 7: **Style transfer.** We manipulate an image (a) to match the artistic style of (b), producing the result (c).

losses are calculated between the input image (the same as content image or a random noise) and the reference image (style/content).

1. (**1 point**) First, we will implement the *content loss*. This loss encourages the generated image to match the scene structure of the content image. We will implement this loss as the squared ℓ_2 distance between two convolutional feature maps. Given a feature map of input image F^x and the feature map of content image F^c, both of shape (C, H, W), the content loss is calculated as follows:

$$\mathcal{L}_c = \sum_{c,i,j} (F^c_{c,i,j} - F^x_{c,i,j})^2. \tag{4}$$

2. (**2 point**) Next, we will implement the *style loss*. This loss encourages the texture of the resulting image to match the input style image. We compute a weighted, squared ℓ_2 distance between Gram matrices for several layers of the network.

   The first step is to calculate the Gram matrix. Given a feature map F of size (C, H, W), the Gram matrix $G \in \mathbb{R}^{C \times C}$ computes the sum of products between channels. The entries k, l of the matrix are computed as:

$$G_{k,l} = \sum_{i,j} F_{k,i,j} F_{l,i,j}. \tag{5}$$

   The second step is to compare the generated image's Gram matrix with that of the input style image. Define the Gram matrix of input image feature map and style image feature map of at the l^{th} layer as $G^{x,l}$ and $G^{s,l}$, and the weight of the layer as w^l. Style loss at the l^{th} layer is:

$$L^l_s = w^l \sum_{i,j} (G^{x,l}_{i,j} - G^{s,l}_{i,j})^2, \tag{6}$$

   where w^l is the weight of layer l. The total style loss is a sum over all style layers:

$$\mathcal{L}_s = \sum_l L^l_s. \tag{7}$$

   Please complete the following tasks:

   - Implement the Gram matrix without a loop or comprehension.
   - Based on your Gram matrix, write down the code for style loss.

3. (**1 point**) Finally, we will try varying the layer used to compute the content loss. Try several different layers $l = 0, 1, 2, ..., 12$ and report which one gave you the best qualitative results.

4. (**Optional, 0 points**) Upload any of your own images or try out other images in the datasets and perform style transfer. We encourage you to post your art to the Piazza thread!

Acknowledgements. Part of the homework dataset and the starter code are taken from a previous EECS 498 class taught by Justin Johnson, which itself was adapted from CS231n at Stanford University by Fei-Fei Li, Justin Johnson and Serena Yeung. Please feel free to similarly re-use our problems while crediting these other classes.

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

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

[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.

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.