Cornell University
CS 5670: Introduction to Computer Vision
Fall 2025. Instructor: Andrew Owens

Problem Set 4: Image Generation

Posted: Friday, October 17, 2025	**Due:** Monday, November 3, 2025

Please submit your Colab notebook to Gradescope as a `.pdf` file. Please convert your Colab notebook to PDF. For your convenience, we have included the PDF conversion script at the end of the notebook.

Colab Notebooks:

- Problem 4.1: https://drive.google.com/file/d/1BlUdgwFBRdWsNRIC6D7vJWEAnhxig0gm
- Problem 4.2: https://drive.google.com/file/d/1EcvS0SML0Ub2y8Lwq0bFgflEzWspySew

Please see **GPU Tips** at the end of this document to better understand how to efficiently use GPUs for this homework set (and all future homeworks that require GPUs). **tldr:** your code will run *very* slow if you run out of GPU hours early and you have to train your models on CPUs. Please follow the tips we provide to avoid this! We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 4.1 *(12 points) Generative Adversarial Networks*

Colab Notebook: https://drive.google.com/file/d/1BlUdgwFBRdWsNRIC6D7vJWEAnhxig0gm

In this problem, we will implement an image-to-image translation program based on pix2pix [3]. We will train the pix2pix model on the *edges2shoes* dataset [3, 6] 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 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.

1. (**6 points**) Let us first 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 2). We have included a toy U-net example in the Colab notebook.

   Note: We use the notation Ck to 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

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.

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

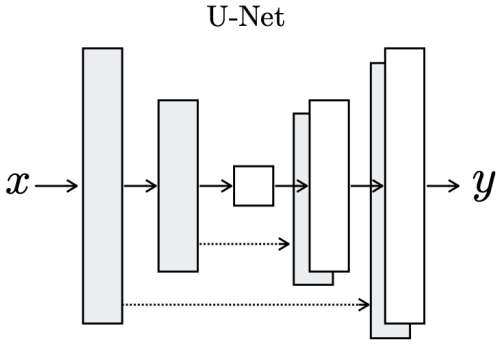

Figure 2: U-net architecture.

(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.

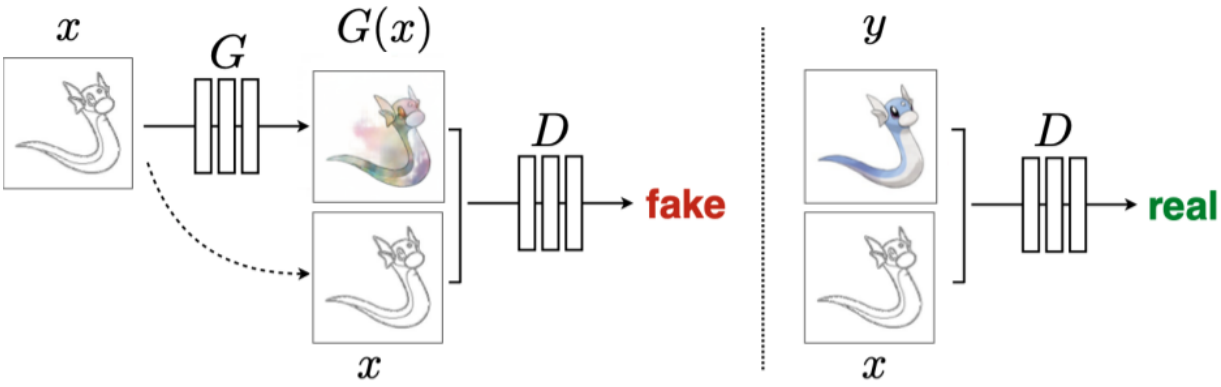

Figure 3: Conditional GAN for image translation.Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real edge, photo tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map. The *Pokémon* images shown here come from *Pokémon Images Dataset* [1, 2].

2. (**6 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 (we call this the **L1 only** model), the other with Equation 3 and $\lambda = 100$ (we call this the **L1 + cGAN** model). Train the L1 only model for 10 epochs and the L1 + cGAN model for 20 epochs. You are welcome to train longer to potentially obtain better results.

Please complete the following tasks:

- Complete all the incomplete code blocks in the Colab notebook.
- In the specified text cell of the Colab notebook (marked by TODO at the end of Part 2.1 in the notebook), comment on the difference between the translated images obtained from the L1 only and L1 + cGAN models.
- Show the history of the generator and the discriminator L1 loss vs. iteration for the L1 only model in 2 separate plots.
- Show the history of the generator's BCE and L1 losses and the discriminator's loss vs. iteration for the L1 + cGAN model in 3 separate plots.
- In the specified text cell of the Colab notebook (marked by TODO at the end of Part 2.2 in the 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 L1 + cGAN model?

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

3. (**Optional**) 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. An example sketch and its corresponding synthesized shoe image are plotted in Figure 4.

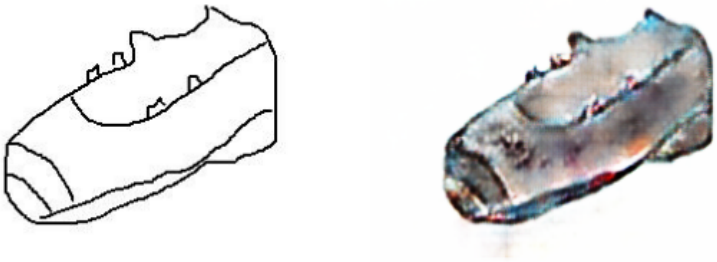

Figure 4: Left: An example sketch. Right: the synthesized shoe image.

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.

Problem 4.2 ***(18 points)*** *Diffusion Models*

Colab Notebook: https://drive.google.com/file/d/1EcvS0SML0Ub2y8Lwq0bFgflEzWspySew

In this problem, we will explore using a pre-trained diffusion model (DeepFloyd IF [4]) to perform image generation and editing. DeepFloyd IF is a two stage model trained by Stability AI. Here, the first stage produces images of size 64×64 and the second stage takes the output of the first stage and generates images of size 256×256.

Below, we list the questions to be solved for this problem. Please refer to the Colab notebook for detailed instructions on how to solve these problems.

Figure 5: Example of a *visual anagram*. When flipped upside down, the image changes its appearance.

- (**1 points**) Part 1: Implementing the forward process.
- (**1 points**) Part 2: Classical denoising.
- (**2 points**) Part 3: One step diffusion denoising.
- (**2 points**) Part 4: Iterative diffusion denoising.
- (**1 points**) Part 5: Diffusion model sampling using your functions.
- (**2 points**) Part 6: Classifier free guidance.
- (**1 points**) Part 7.1: Image to image translation.
- (**1 points**) Part 7.2: Translating your own images.
- (**1 points**) Part 7.3: Text conditioned image editing.
- (**2 points**) Part 8: Inpainting.
- (**2 points**) Part 9: Visual anagrams.
- (**2 points**) Part 10: Hybrid images.

Acknowledgements. The GAN problems were created as part of UMich's EECS 442 by Andrew Owens. The diffusion problems were created as part of a joint effort by UMich EECS 442 and UC Berkeley's CS 180/280A, by Daniel Geng, Hang Gao, Ryan Tabrizi, Liyue Shen, Andrew Owens, and Alexei A. Efros.

1 GPU Tips

For the remainder of the class, we will be using GPUs with Google Colab. We use Colab, since we believe that it is the best possible option (given the fact that high quality GPUs are very expensive). However, there are a number of challenges due to the limits that Colab puts on GPU usage for free accounts. To help you get some information on this and make the most use of the free limits we have compiled a list of best practices to follow.

Colaboratory or simply Colab is part of the Google suite of products. Put simply, Colab is a free Jupyter notebook environment that runs entirely in the cloud. You can read more about it here: https://colab.research.google.com/. We use Colab with Python environment and to speed up model training. We will be using a very important feature of Colab which is GPU access.

Colab usually provides T4 GPU for free accounts. Very rarely you'll get a A100 or L4, which are much faster. Colab has time and usage limits for GPU access and the rules for these limits are vague so we have to be careful using them. The general observation is that you start off with a fixed (4 to 8hr, could be lower) bank. Once you exhaust that, you will not longer be able to use GPU for upto 24hrs. Once the counter resets, you get lower hours of usage each time. So how do we maximize our free GPU resources?

(1) **Use CPU for debugging**

The majority of the implementation in most of the assignments can be completed in CPU; you should only switch to GPU once you are confident about your code and ready to train your finals models. The notebook we give you might connect to a GPU runtime by default. You can change between CPU and GPU instances using Runtime → Change Runtime Type; this resets the notebook backend, so you'll need to rerun all cells after switching instance types. You will also need to change `device = torch.device('cuda')` to `device = torch.device('cpu')` Please make sure you don't have any `.cuda()` statements but rather `.to(device)`

Also, please note that **you do not need the GPU for problems 2.1 and 2.2, so make sure to run in CPU-only mode for these problems.**

(2) **Use other Google accounts**

If you have multiple Google accounts, please note that each one has a separate limit.

Enable cell execution notifications: Colab can notify you of completed executions even if you switch to another tab, window, or application. You can enable it via Tools → Settings → Site → Show desktop notifications (and allow browser notifications once prompted) to check it out. This is helpful when training models so you don't leave GPU connected and idle for long times.

(3) **Debugging tips for GPU**

```
RuntimeError: CUDA device-side assert triggered
```

Do not rely on the line where this error is raised. Even the error traceback says this:

RuntimeError: CUDA error: device-side assert triggered CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect. For debugging consider passing `CUDA_LAUNCH_BLOCKING=1`

The stack trace might be incorrect. For better debugging, re-run your code as

```
CUDA_LAUNCH_BLOCKING=1 python script.py
```

or add

```
import os
os.environ["CUDA_LAUNCH_BLOCKING"] = 1
```

at the top of your code (temporarily) for debugging. The traceback you get now will point you to the actual line where the error occurs.

(4) **Non-Colab options**

We highly recommend using Colab. However, there are several other options.

If you have a GPU in your local machine and can set it up to use within Jupyter notebooks, you can use it to run the notebooks. In our experience, this approach is also less reliable, so we recommend using Colab when possible. Also, since the problem sets are designed for Colab, running them here will require modifications. You can also connect Colab to a local runtime with GPU. Instructions here: https://research.google.com/colaboratory/local-runtimes.html. Please note that the instructors will not be able to help debug environment setup and code issues on non-Colab systems.

(5) **Colab Pro**

Google offers paid premium upgrades: Colab Pro and Pro Plus which are $10/month and $50/month respectively. These upgrades have access to faster GPUs and for longer. If you need more access, you can consider these options. All assignments will be thoroughly tested to ensure that it can completed within the limits of free Colab resources.

References

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

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

[3] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[4] StabilityAI. Deepfloyd if. https://github.com/deep-floyd/IF.

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

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