Artificial Intelligence for the Media
Mini-Project Brief
By Marissa Beaty

The goal of this project was to train a Pix2Pix model to take an image of a building that has been edge detected and have it produce a realistic likeness of that building. To do so, the Pix2Pix model will train on a paired image set of architectural images from Pinterest, and those same images run through a Canny edge detection model, saved as one paired input that the model would split and train on. Four different Pix2Pix models were selected and adjusted to attempt training, however, all four failed to produce generative images. As a final experiment, these images trained using a CycleGAN model, which yielded preliminary but promising results. This paper will briefly discuss the aspects of each model and provide insight into the project's failures and successes.

The Pix2Pix model in the first notebook was the first attempt at Pix2Pix training. This model was based on the paper by Phillip Isola et. al and interpreted by Aniket Maurya. All four Pix2Pix models utilize a U-Net training structure. Unique to this model was the technique splitting the paired image into the real and condition (edge) images and how those images were translated into a Tensor readable by the program. Though several other aspects of the model needed alterations to fit the dataset and correct errors, this was the largest endeavor. The process above - splitting and tensoring the images - caused significant errors when read by the DataLoader. The DataLoader was reading the images by height, width, and RGB, but a Pix2Pix model trains on images structured as RGB, height and width. Attempts to fix this error include: tensoring the image before splitting, swapping positions of the values using swapaxes and unsqueeze, rewriting the class that reads in the dataset, etc. Unfortunately, these changes only created new errors or resulted in the original error.

The second notebook includes a model adapted from OpenCV. I selected this model because it removed classes, favoring a series of functions as its structural foundation. My theory being the difficulty with reading the images correctly in the previous model was due to an error with the DataLoader reading and cycling through the class that defined and created the tensored images. Unlike the previous model, this model did get to the training stage, however, when beginning training, it produced the following error message: "ToTensor.__init__() takes 1 positional argument but 2 were given" referencing the DataLoader. I understood, then, that the issue was not due to the class but rather to the structure of the Pix2Pix model reading and splitting the paired image. I attempted to fix the error by pulling the matched images from separate folders rather than inputting and splitting a single image. The further I got in this transition, the more I realized it would require a complete restructuring and reconfiguring of the model. Given the available timeframe, this was not a feasible endeavor.

The next model I attempted was that written by the creators of the Pix2Pix model: the research team of Phillip Isola, et al. This model is within a GoogleColab notebook. Though the colab notebook worked initially, I again found an issue when uploading my dataset into the model, particularly with the inability to find the directory that housed my data and download the model necessary to train the data. Though the error seems relatively simple to resolve, since the model is not within the Colab notebook, it was difficult to determine what could be the root cause. The limited solutions found during research proved unsuccessful.

In one last attempt to utilize a Pix2Pix model, I adopted the model posted by Aryan Esfandiari on a personal blog, documented in the fourth notebook. This model implemented

Pytorch Lightning and established a different way of splitting the image into a real and condition set. In training this model, however, I came upon a new error only halfway through the code that stated, “Can’t pickle local object ‘SketchDataset.__init__.<locals>.<lambda>’.” This error seemed straightforward, requiring me to alter my object from a local to a global object. However, despite doing this, the error remained. I had attempted to bypass Pickle by implementing Dill but without success. Much like the errors produced with the previous Pix2Pix models, resources on this error were hard to find and implement, and unsuccessful in resolving the errors.

After four failed attempts at training a Pix2Pix model, I became curious towards whether a CycleGAN might produce similar difficulties. The appeal of a CycleGAN model is that it does not require perfectly paired images, which seemed to be the failing factor of the Pix2Pix models. I utilized a CycleGAN model published to Kaggle by SW-Song. Due to time constraints, I trained this model with only 800 total images, 400 edge images, and 400 real images (or condition and real as referred to in the Pix2Pix model), and only five epochs compared to a recommended 20. Having tweaked the code reading in the datasets, the model began training and produced 25 images for each epoch and batch after approximately 23 hours.

The results of the CycleGAN model are not perfect. Though the model did improve over time, the model was designed to train on a dataset of 7,000 to 10,000 images, utilizing a larger GPU than I had available. I expect the limited size of my dataset and the reduced number of epochs to be the foundational cause for the imperfect images produced by the model. Despite this, the CycleGAN model is a clear choice over the Pix2Pix model. Where all Pix2Pix models failed was in reading the paired image into the model. The CycleGAN bypasses this by not only pulling from separate image folders but not requiring these images to be pulled into the model as pairs. This seemingly simple structural change is what made all the difference when testing and training the models.

Despite the many failures of the Pix2Pix model, the process involved with researching errors, reconfiguring classes, and manipulating these models provided a deep understanding of the networks structure, so much so that I feel confident attempting the construction of a model like this on my own, something I could not have stated when this project began. For that reason, I deem the project a success, regardless of the failed and imperfect outputs.

Works Cited

Pix2Pix Model used in Notebook 1
Aniket Maurya (2021) Pix2Pix – Image to Image Translation with Conditional Adversarial Network [Source Code]. https://librecv.github.io/blog/gans/pytorch/2021/02/13/Pix2Pix-explained-with-code.html.

Pix2Pix Model used in Notebook 2
Aditya Sharma (2021) Pix2Pix: Image-to-Image Translation in Pytorch and Tensorflow [Source Code]. https://learnopencv.com/paired-image-to-image-translation-pix2pix/.

Pix2Pix Model used in Terminal for Trial 3
Phillip Isola and Jun-Yan Zhu et al, (2017) CycleGAN and Pix2Pix in Pytorch [Source Code]. https://colab.research.google.com/github/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/pix2pix.ipynb#scrollTo=9UkcaFZiyASl.

Pix2Pix Model used in Notebook 4
Aryan Esfandiari (2021) Pix2Pix with Pytorch and PytorchLightning [Source Code].
https://www.aryan.no/post/pix2pix/pix2pix/.

CycleGAN Model used with successful results
SW-Song (2021) CycleGAN Tutorial from Scratch: Monet to Photo [Source Code].
https://www.kaggle.com/code/songseungwon/cyclegan-tutorial-from-scratch-monet-to-photo.