Image to Image Translation of Ultrasound Images using Pix2Pix

Anonymous CVPR submission

Paper ID 1234567

1. Introduction

The motivation of this project is to use image to image translation to design a program that generates ultrasound images given an outlined mask. This project potentially could be used as a real-time program for medical professionals to have a second opinion on what they are seeing and analyzing. During pregnancy, the gestational age (GA) of a fetus is a critical piece of information for doctors. The GA gives obstetric care doctors a good indicator regarding the health of the fetus's growth. Combined with other biometrics such as the fetal head circumference and estimated weight, doctors can give more reliable care for the mother and the baby. However, these ultrasound images have to be individually processed and analyzed by the doctors and nurses. Having a second opinion could serve a pivotal role in providing better care for mothers and their babies. Using Pix2Pix, an image to image translation architecture, we attempt to train a generator to produce ultrasound images from a mask, and a discriminator to discern which ultrasound images are real. Due to the rise of low-cost hardware, ultrasound imaging is playing a major role in global public health, especially in developing countries and resource-constrained areas. Ultrasound technology is relatively inexpensive and plays a huge role in the fetus's health during pregnancy. However, it takes a long time to train qualified professionals to operate and interpret the information and images gathered from ultrasound technology. Today, there is high demand for developing new AI methods to effectively train practitioners and interpret images.

2. Related Work

2.1. Pix2Pix

Pix2Pix is both a pared and unpaired image to image translation model, utilizing tensorflow and pytorch. The project is comprised of a Generator model and a Discriminator model. The Generator is a conditional generative adversarial network (cGAN), meaning an input must be given to the generator (i.e. instead of giving it nothing). Being essentially a UNET, an image is preprocessed in dataloading then goes through a series of down convolutions, batch normalizations, and activation functions, all the while saving the gradients. After the bottleneck layer, the image tensor goes through a series of upconvolutions where the gradients of the down layer are concatenated in 'skip connections'. This preserves some of the form of the original image while pulling all the other useful information out in the process. Figure 1 shows the detailed architecture.

After an image tensor is generated using this UNET-like model, it is passed to the Discriminator. The Discriminator attempts to determine whether the image it has been given is real or not. Being a more streamlined model, the images are concatenated and run through a series of convolutions, sequential layers, activation functions, batch normalization, and zero padding. Figure 2 shows the detailed architecture. Using what the authors of Pix2Pix call a "patch-GAN" attempts to classify each NxN patch as being a real or fake image. In running this discriminator convolutionally over the image, the output of the discriminator becomes the average of all the convolutional responses. The Generator uses a sigmoid cross-entropy loss of the generated images and an array of ones. L1 loss is also incorporated with the generated image and the test image so they become more structurally similar. The Discriminator also uses a sigmoid cross-entropy loss on the real images and an array of ones. This 'real loss' is coupled with another sigmoid cross-entropy loss, 'generated loss', on the generated images and an array of zeros. The total loss of the Discriminator is the sum of the real loss and the generated loss. Adam optimizer is used for both models.

3. Methods

3.1. Image Preprocessing

Pix2Pix utilizes tensorflow datasets to more efficiently and robustly preprocess images. First, images are resized from shape (256, 256) to (286, 286). Then a random crop of shape (256, 256) is taken and randomly flipped horizontally. This helps both the models become more robust by introducing slight variance to data. Finally since we want values to be normalized between [-1, 1], we load our data into a tensorflow dataset after applying a normalization function.

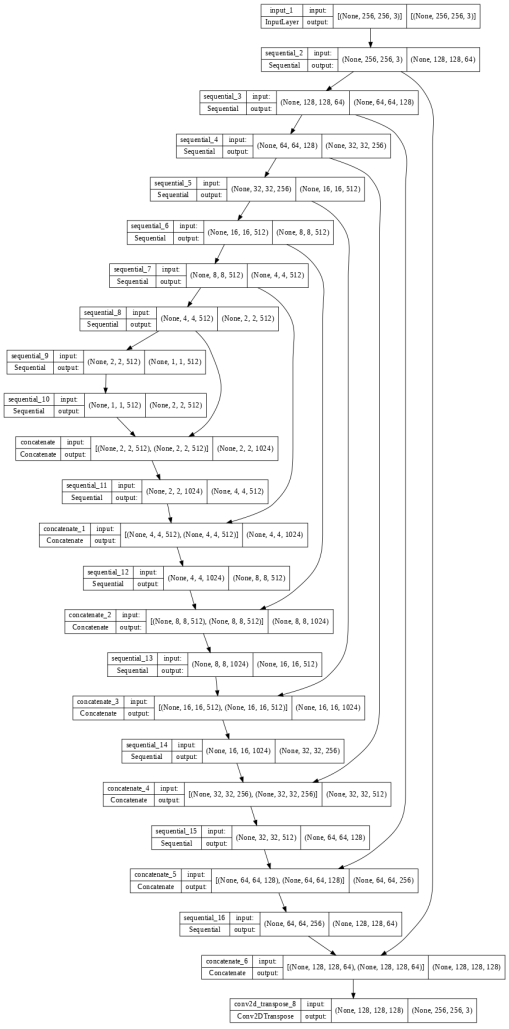

Figure 1. Generator U-net Architecture

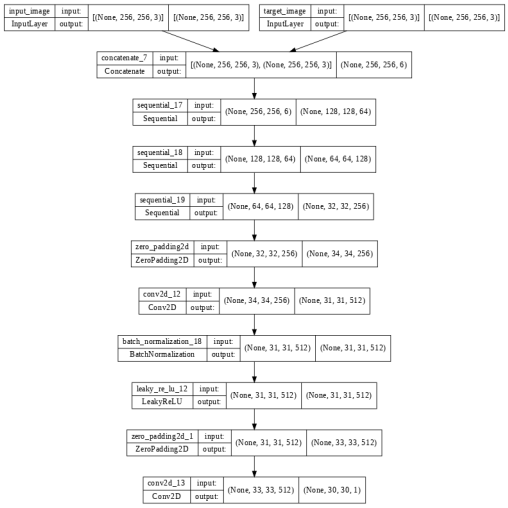

Figure 2. Discriminator Model Architecture

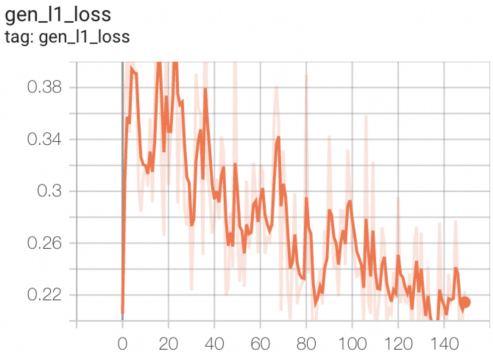

Figure 3. Generator Loss

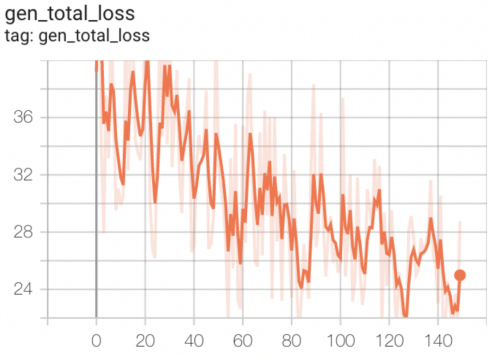

Figure 4. Discriminator Loss

3.2. Training

Training begins by passing the the input masks to the generator, which returns the generated images. These generated images, along with the corresponding preprocessed input images and target images, are then passed to the Discriminator. The losses for each model are then calculated by the methods listed earlier. After calculating the gradients of these losses with respect to the Generator and Discriminator, they are applied to the optimizer. we ran our training function over a minibatch of our data due to complications with colab and GPU usage. Figures 4 and 5 highlight the training processes for each model.

4. Experiments

Most of the experimentation was focused around dataloading and data management. The python Pix2Pix code supports any data type, but acknowledges that alterations must be made to accommodate data that doesn't strictly fit their format. The datasets they used are single .jpg images where the left half is the mask and the right half is the real image. We were unable to find a good way of converting our individual files into singular concatenated images. We experimented with a few different dataloaders includ-

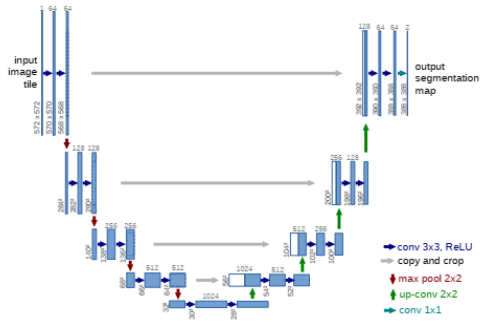

Figure 5. U-net Architecture

Figure 6. Comparison

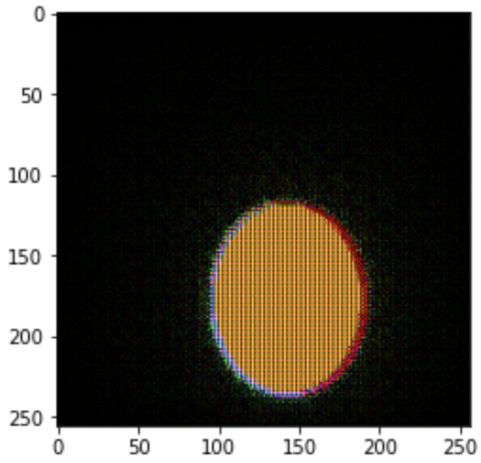

Figure 7. Generator Overlay

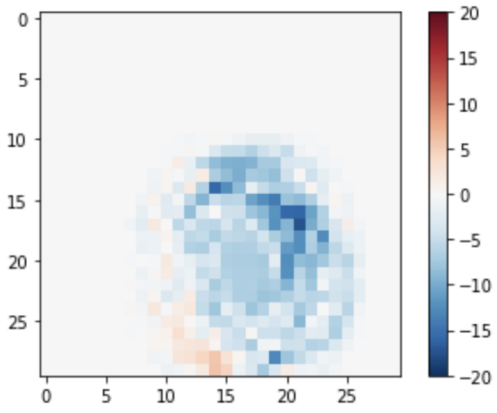

Figure 8. Discriminator Heat Map

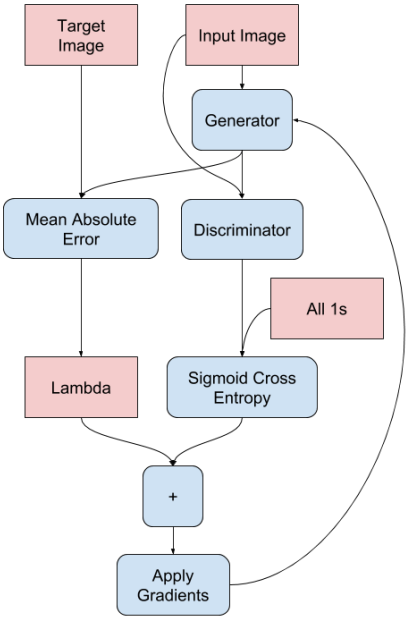

Figure 9. Generator Training Pipeline

ing tf.data.Dataset and DataLoader, ultimately settling with a python list containing the EagerTensors of each instance of an image.

5. Conclusions

Ultimately our implementation of pix2pix with ultrasound images yielded decent results as our losses continually fell throughout the training. That said, no definitive conclusions can be made as this dataset is incredibly small and our program was not as robust as it should have been.

6. GitHub Link

GitHub Link

7. Contribution

Riley - Worked on all the code, and most of the report.

Sean - Worked on formatting and preparing presentation slides, and fine tuning the training function

Michael - Worked on formatting and preparing presentation slides, and coming up with dataloading workarounds.

References

Code Reference and Data: https://github.com/Pankaj1357/HC18-Grand-Challenge

Pix2Pix colab link: https://colab.research.google.com/github/junyanz/pyto
CycleGAN-and-pix2pix/blob/master/pix2pix.ipynb

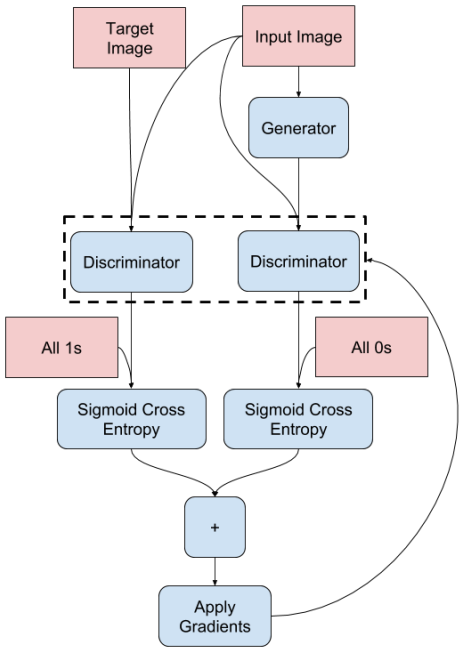

Figure 10. Discriminator Training Pipeline

Unet: Olaf Ronneberger, arXiv:1505.04597

Optimization Strategies:Irwan Bello, William Fedus, Xianzhi Du, Ekin D. Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, Barret Zoph, https://arxiv.org/abs/2103.07579.pdf

pix2pix github: https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

Pix2pix Colab Tutorial: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/pix2pix.ipynb

Paper for pix2pix: https://ieeexplore.ieee.org/document/8681706

Random Augmentation: https://arxiv.org/abs/1909.13719