Image Denoising with cGAN

Using pix2pix architecture

Aim

Use cGAN in image denoising application.

- important step in image preprocessing
- noise from various of sources

Interpretation of noisy images

$$L'(x, y) = L(x, y) + N(x, y) \qquad (1)$$

where:

L':	received (noisy) image,
L:	orignal signal (expected image),
N:	random noise,
x, y:	pixel coordinates.

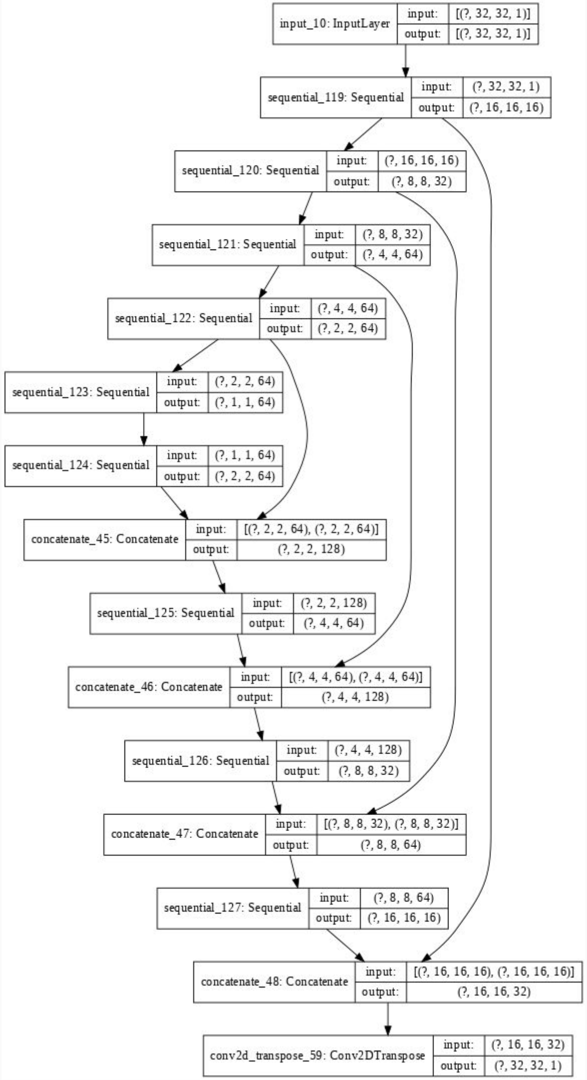
cGAN - Conditional Generative Adversarial Network

cGAN - what is it?

- extension to GAN
- generating new examples

In our case **generate image without noise from image with noise**.

We used **pix2pix** architecture.

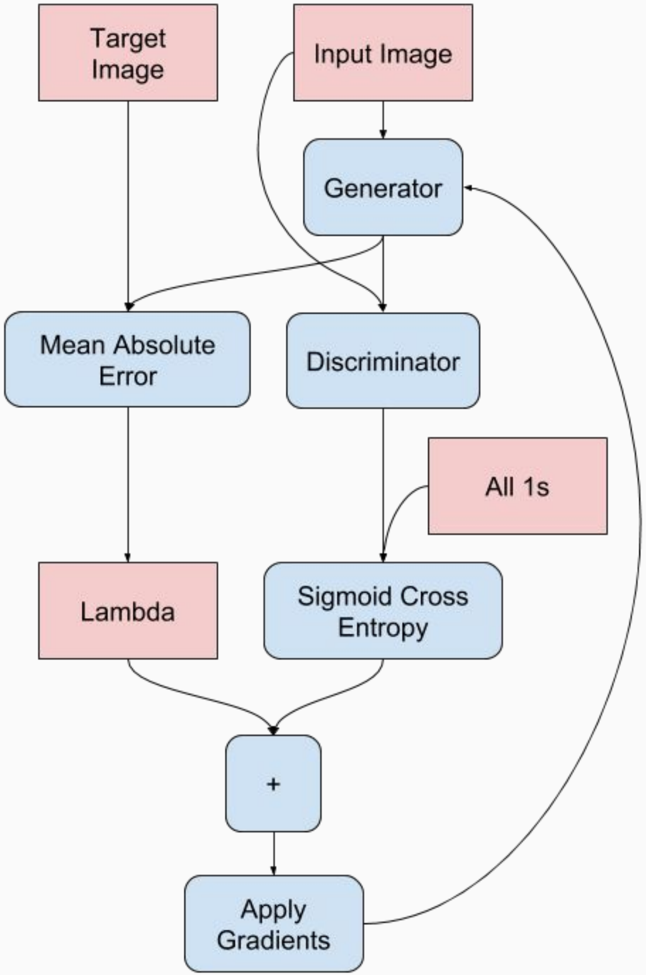


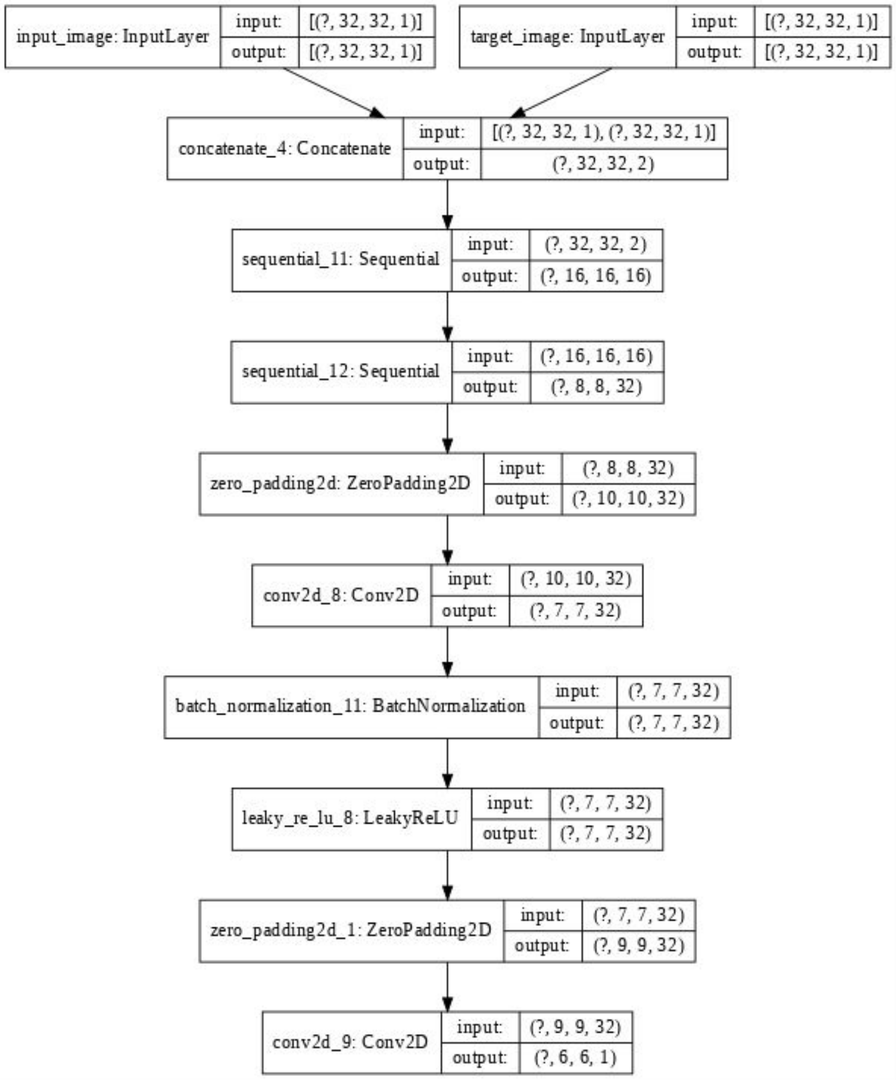
Generator:

- The architecture of generator is a modified U-Net.
- Each block in the encoder is (Conv -> Batchnorm -> Leaky ReLU)
- Each block in the decoder is (Transposed Conv -> Batchnorm -> Dropout(applied to the first 3 blocks) -> ReLU)
- There are skip connections between the encoder and decoder (as in U-Net).

Generator loss

- It is a sigmoid cross entropy loss of the generated images and an array of ones.
- L1 loss which is MAE / SSIM between the generated image and the target image.
- This allows the generated image to become structurally similar to the target image.





Discriminator:

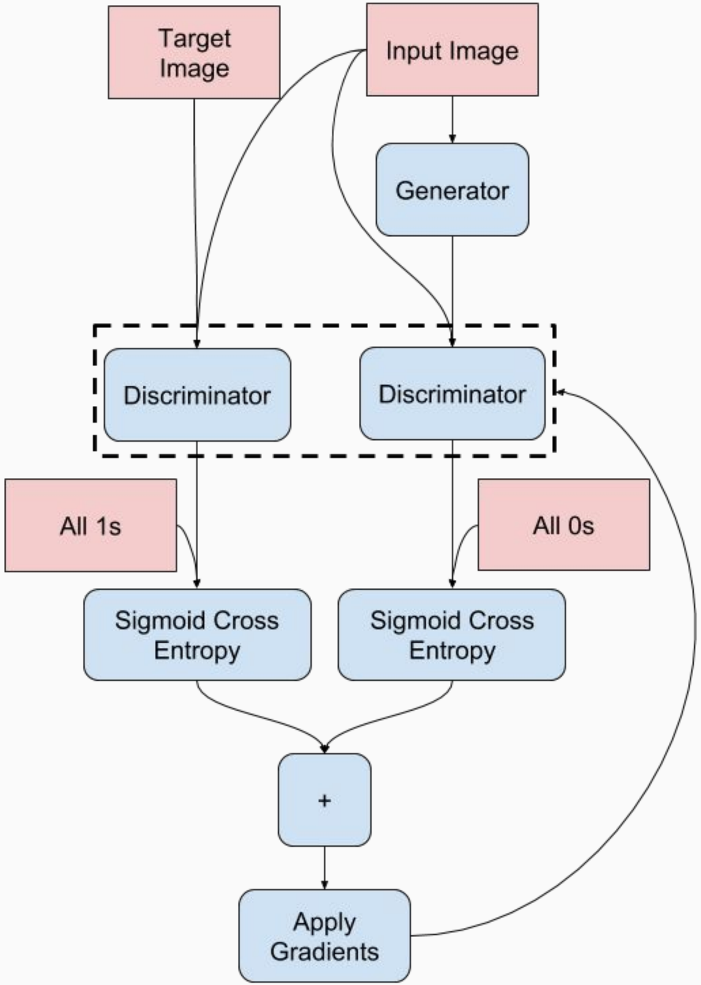
- The Discriminator is a PatchGAN.
- Each block in the discriminator is (Conv -> BatchNorm -> Leaky ReLU)
- The shape of the output after the last layer is (batch_size, 6, 6, 1)
- Each 6x6 patch of the output classifies a 14x14 portion of the input image (such an architecture is called a PatchGAN).
- Discriminator receives 2 inputs

Discriminator loss

- **real_loss** is a sigmoid cross entropy loss of the real images and an array of ones(real images)

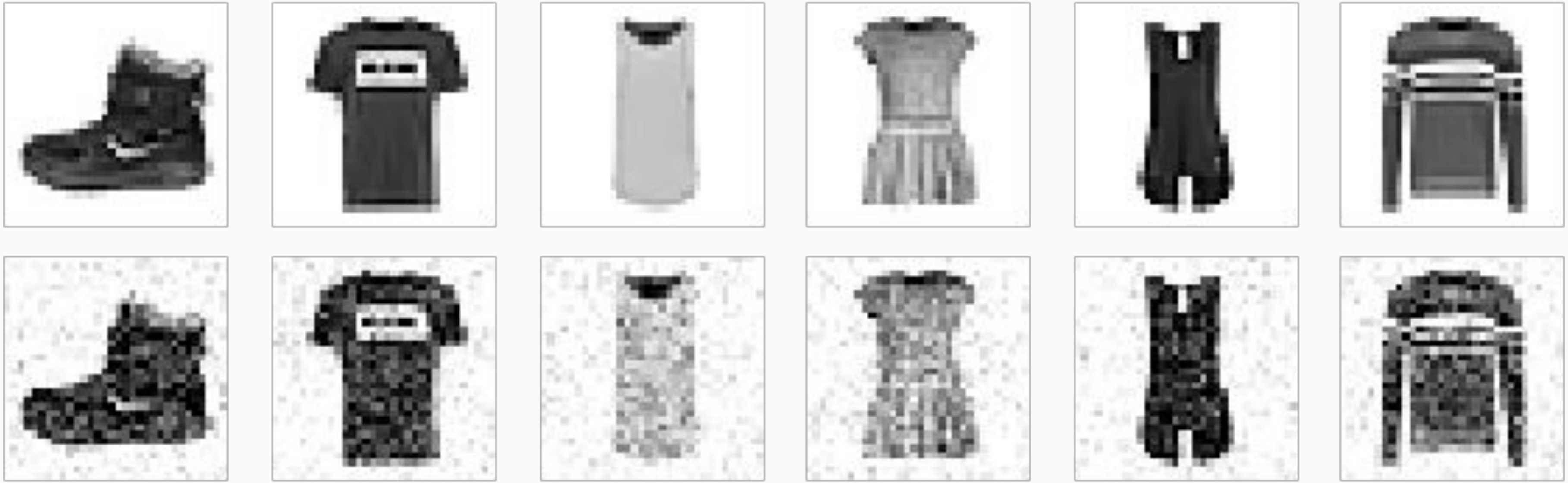
- **generated_loss** is a sigmoid cross entropy loss of the generated images and an array of zeros

- **total_loss** is the sum of real_loss and the generated_loss



Data set - fashion MNIST

- Original data - 28x28 with padding of 2 = (32x32)
- Noisy images:
  - Gaussian noise
  - var <30; 50>



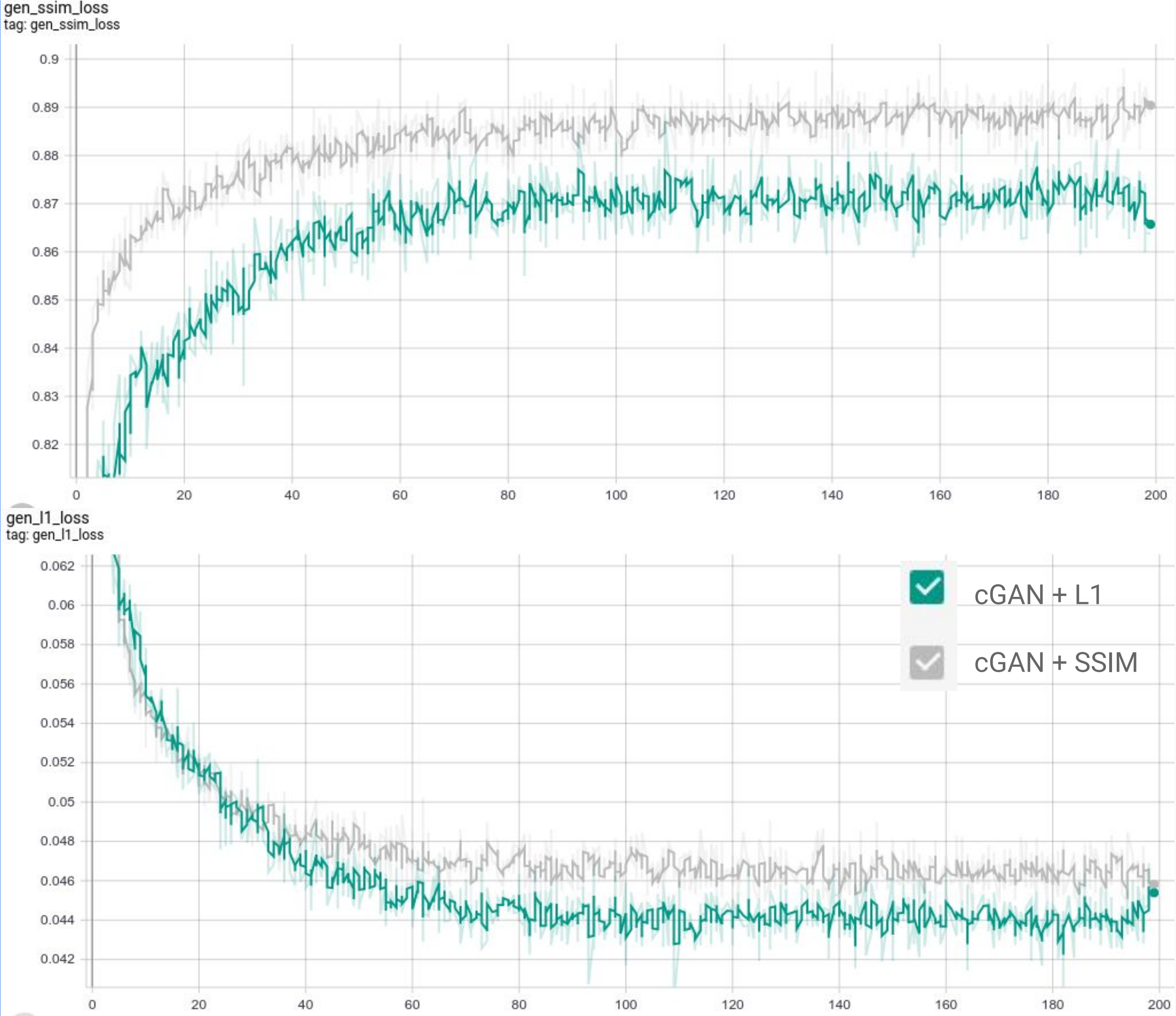
Train the pix2pix

- For each example input generate an output.
- The discriminator receives the input_image and the generated image as the first input. The second input is the input_image and the target_image.
- Next, we calculate the generator and the discriminator loss.
- Then, we calculate the gradients of loss with respect to both the generator and the discriminator variables(inputs) and apply those to the optimizer.
- Then log the losses to TensorBoard.

Training with different loss functions:

- SSIM - Structural similarity

- L1 - MAE

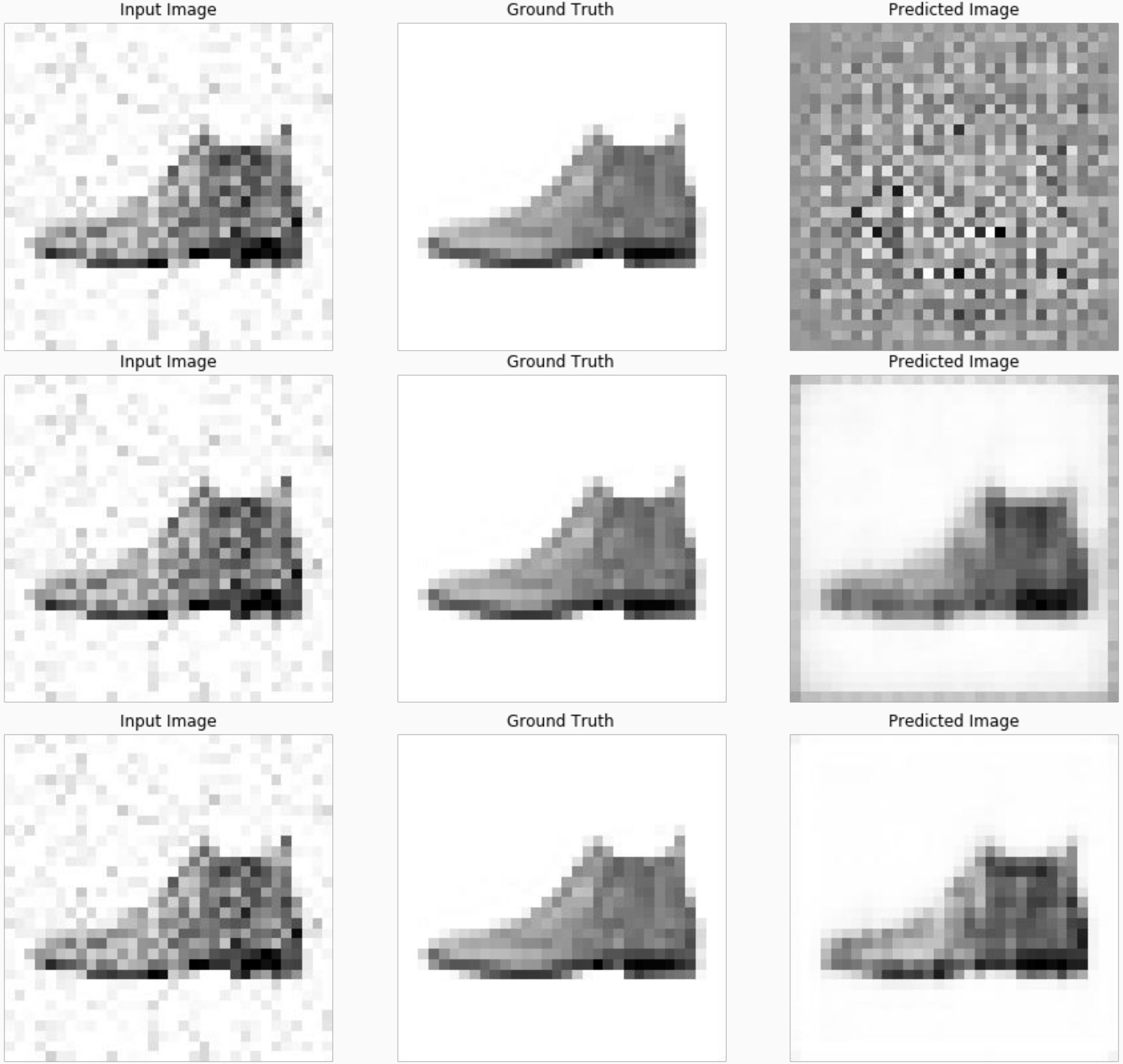


Results

0 epoch:

2 epoch:

4 epoch:



Train Set - different loss functions

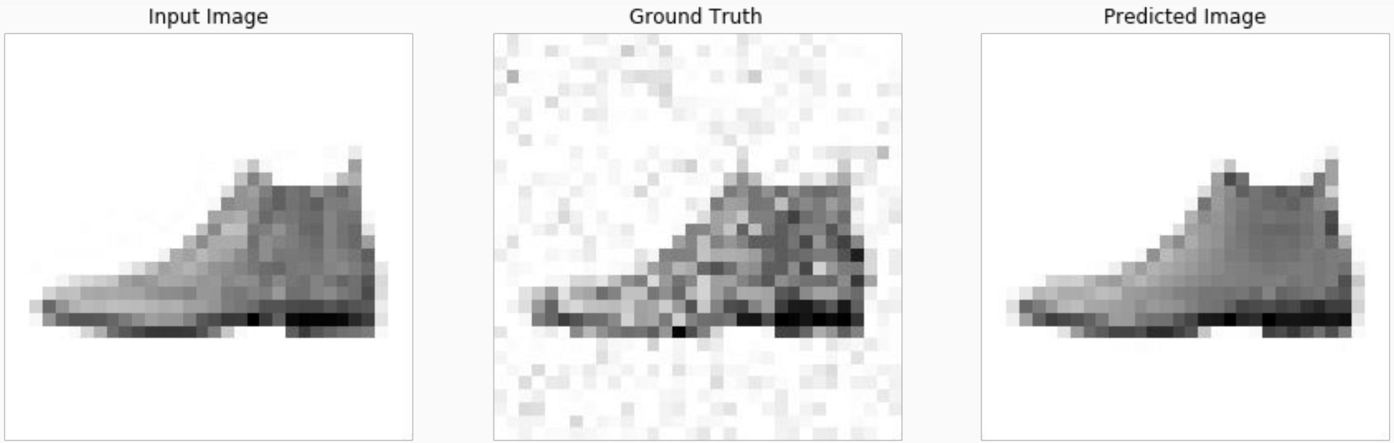
gen loss	L1 (MAE)	SSIM	PSNR
Noisy Images	0.123	0.681	20.55
cGAN + SSIM	0.046	**0.888**	26.12
cGAN + L1	**0.045**	0.865	**26.19**
Ground Truth	0.0	1.0	Inf

Test set - different loss functions

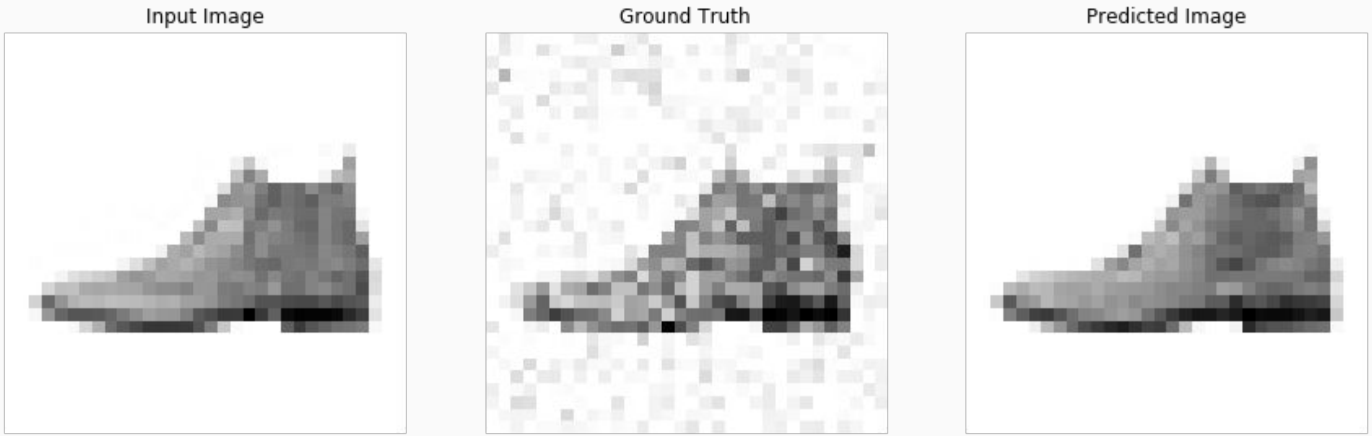
gen loss	L1 (MAE)	SSIM	PSNR
Noisy Images	0.123	0.681	20.54
cGAN + SSIM	0.048	**0.871**	25.86
cGAN + L1	**0.046**	0.861	**26.09**
Ground Truth	0.0	1.0	Inf

Final results

SSIM:



L1:



Ideas for improvements

- More sophisticated noise
- More various image types
- Train on images with bigger size and RGB
- Evaluate on some real set of images

Thank you!