Image Repair Using Deep Convolutional Generative Adversarial Networks

Martin Trpceski
Department of Electrical and Computer Engineering
University of Rochester
Rochester, United States
mtrpcesk@u.rochester.edu

Sean Gleason
Department of Electrical and Computer Engineering
University of Rochester
Rochester, United States
sgleason6@u.rochester.edu

***Abstract*—Historical images that have been subject to decay and neglect often have suffered considerable damage. In addition, old imaging techniques leave many other imperfections. This has led to services that are specifically designed around image repair. However, these services can be quite expensive. Generative adversarial networks (GANs) are a new type of neural network that have been used in a wide variety of generation tasks, including image repair. GANs that are used for image generation tasks are called Deep Convolutional GANs (DCGANs). A custom DCGAN architecture was created and then compared to a modifier version of the StyleGAN architecture. The custom model was trained on the Imagenet dataset, where image processing was used to simulate the damages that could be caused by age on old photos.**

***Index Terms*—DCGANs, Imagenet, StyleGAN**

I. Introduction

Through time, poor maintenance, and poor image quality, historical images are often left in poor condition. For that reason, there are countless services that promise to restore historical photos. However, these services can be expensive and out of reach for many who might use them.

Generative AI in recent years has opened opportunities for wide ranges of image modification and diffusion tasks, including for image restoration and repair. Deep Convolutional Generative Adversarial Networks (DCGANs) are one of the leading edge generative neural network applications.

In this paper, a DCGAN was developed and trained on the ImageNet dataset and compared to a benchmark model. Then, both models were subjectively evaluated on a smaller dataset of old images that lack a ground truth.

II. Methods

A. Datasets

To prepare for training a custom dataset was created including a clean and damaged version of the Imagenette dataset. Both were set to one channel grayscale, resized to 256x256, and normalized. To cause artificial damage a mixture of effects were applied. Gaussian blur was added using the torchvision Gaussian Blur transform, and splotches were applied using the scikit-learn make blobs function. Separate code was written to create random cracks in the images as well as decrease the contrast of the image. With the damage effects applied, the damaged and clean versions of the Imagenet dataset were combined, resulting in 9469 pairs of images ready for training. The same process was done again on the validation images, resulting in 3925 pairs of images to be used for testing the accuracy of the model.

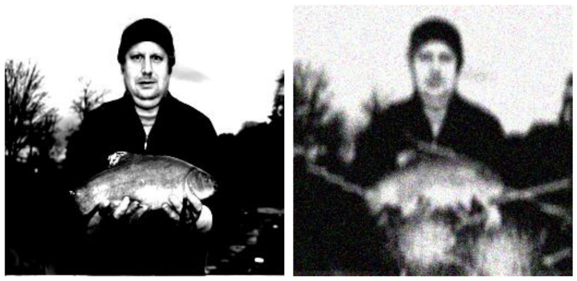


Fig. 1. Example of the training data with the ground truth data (left) and the input data (right)

For a more realistic set of images for final testing, another dataset was developed from the Library of Congress's collection of daguerreotypes [1]. This collection included hundreds of photos from mostly the mid 1800s, the majority of which had damages from either wear over time or the process of taking the photo. These damages included splotches and cracks, some blurring, and low contrast, which is where the set of artificial damage effects originated from.

B. Custom GAN

GANs are one of the leading edge methods for generative AI. They are made up of two individual neural networks: a discriminator and a generator. These models are pitted against each other, with each one constantly improving, where the hope is that the generator eventually triumphs.

The role of the generator is to generate outputs that are most similar to the input data that it has been trained on. For pure generative applications, GANs are used with diffusion, where the input data are often a form of Gaussian noise to act as a sort of seed where the model is trained to generate a specific object [2]. For example, a GAN that can create images of cats on demand. Additionally, for image modification tasks, such as what is done in this project, the input data is often the original

Identify applicable funding agency here. If none, delete this.

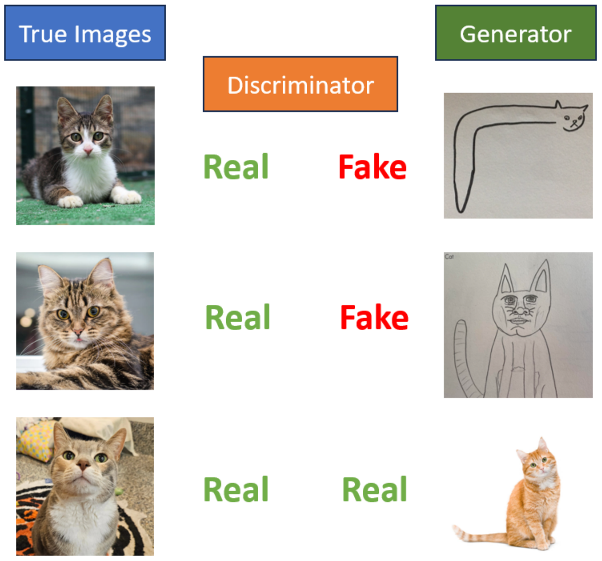


Fig. 2. Overview of the core adversarial aspect of a GAN. Image sources (top left to bottom right): https://www.fourpawsusa.org/our-stories/publications-guides/a-cats-personality https://www.boredpanda.com/funny-poorly-drawn-cats/ https://www.greensboro.carolinavet.com/site/greensboro-specialty-veterinary-blog/2023/03/15/how-to-choose-cat-breed https://www.reddit.com/r/drawing/comments/wcd0lt/my_cat/?rdt=58196 https://www.facebook.com/photo.php?fbid=1030686362439913&id=100064956751160&set=a.325146289660594 https://www.istockphoto.com/photos/domestic-cat

data and the output label is the modified image [3]. The loss of the generator is the pixel-wise loss of the original image with the generated image plus the adversarial loss, which is how well the GAN is able to fool the discriminator. In this example, the adversarial loss is Binary Cross Entropy with Logits, as it is well suited for binary classification tasks.

The goal of the discriminator is to deduce whether or not an input image is a generated image or a real image. The loss of the discriminator is just the adversarial loss.

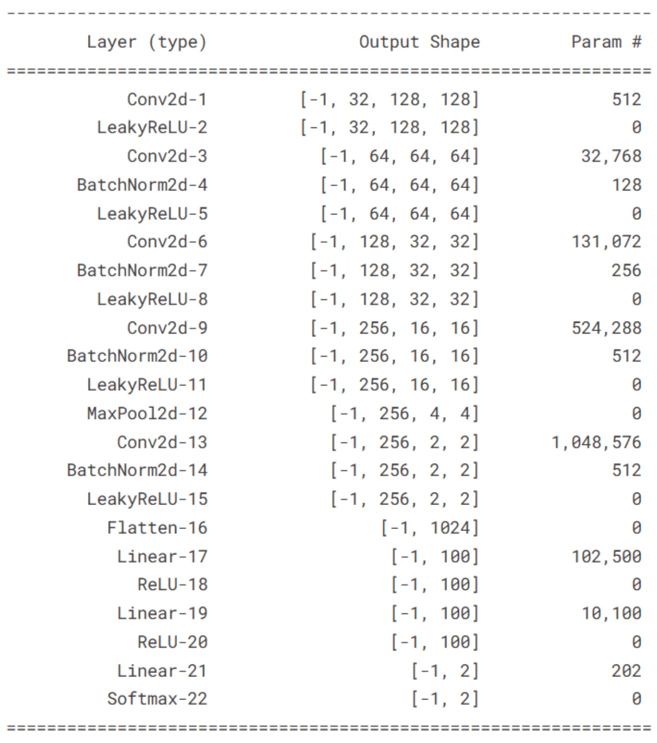
in the training phase, both models have their weights updated, where the goal is to reach a point where the accuracy of the discriminator is around 0.50. At first, the discriminator can easily distinguish fake images from real ones. However, as training continues, the generator get better and better at creating replica images until the discriminator is unable to differentiate.

The discriminator in this project used a simple image classifier that consists of convolutional layers followed by fully connected layers and the Softmax layer.

The generator consists of several convolutional and batch normalization layers designed in such a way that preserves dimensionality of images. That is, the size of input images is the same as images the output images.

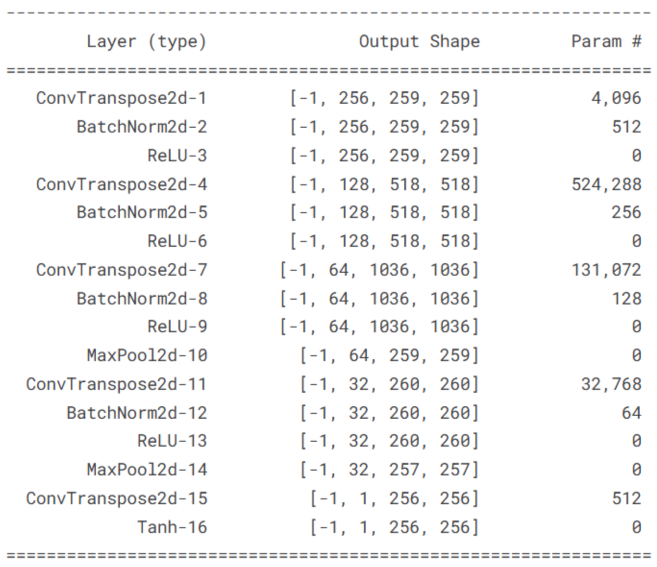
C. *StyleGAN with Latent Space Expansion*

The baseline model that is being used is the model created by Poirier-Ginter at al. in their paper titled ”Robust Unsupervised StyleGAN Image Restoration”. In this paper they train a



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 128, 128]             512
         LeakyReLU-2         [-1, 32, 128, 128]               0
            Conv2d-3           [-1, 64, 64, 64]          32,768
       BatchNorm2d-4           [-1, 64, 64, 64]             128
         LeakyReLU-5           [-1, 64, 64, 64]               0
            Conv2d-6          [-1, 128, 32, 32]         131,072
       BatchNorm2d-7          [-1, 128, 32, 32]             256
         LeakyReLU-8          [-1, 128, 32, 32]               0
            Conv2d-9          [-1, 256, 16, 16]         524,288
      BatchNorm2d-10          [-1, 256, 16, 16]             512
        LeakyReLU-11          [-1, 256, 16, 16]               0
        MaxPool2d-12            [-1, 256, 4, 4]               0
           Conv2d-13            [-1, 256, 2, 2]       1,048,576
      BatchNorm2d-14            [-1, 256, 2, 2]             512
        LeakyReLU-15            [-1, 256, 2, 2]               0
          Flatten-16                 [-1, 1024]               0
           Linear-17                  [-1, 100]         102,500
             ReLU-18                  [-1, 100]               0
           Linear-19                  [-1, 100]          10,100
             ReLU-20                  [-1, 100]               0
           Linear-21                    [-1, 2]             202
          Softmax-22                    [-1, 2]               0
================================================================
```

Fig. 3. Description of the custom GAN discriminator: a binary classifier with a 256x256 grayscale image as the input



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1       [-1, 256, 259, 259]           4,096
       BatchNorm2d-2       [-1, 256, 259, 259]             512
              ReLU-3       [-1, 256, 259, 259]               0
   ConvTranspose2d-4       [-1, 128, 518, 518]         524,288
       BatchNorm2d-5       [-1, 128, 518, 518]             256
              ReLU-6       [-1, 128, 518, 518]               0
   ConvTranspose2d-7     [-1, 64, 1036, 1036]          131,072
       BatchNorm2d-8     [-1, 64, 1036, 1036]              128
              ReLU-9     [-1, 64, 1036, 1036]                0
        MaxPool2d-10       [-1, 64, 259, 259]                0
  ConvTranspose2d-11       [-1, 32, 260, 260]           32,768
      BatchNorm2d-12       [-1, 32, 260, 260]               64
             ReLU-13       [-1, 32, 260, 260]                0
        MaxPool2d-14       [-1, 32, 257, 257]                0
  ConvTranspose2d-15        [-1, 1, 256, 256]              512
             Tanh-16        [-1, 1, 256, 256]                0
================================================================
```

Fig. 4. Description of the custom GAN generator with a 256x256 grayscale image as the input

StyleGAN model with the correct set of hyper-parameters and dataset in order to successfully reconstruct damaged images [3].

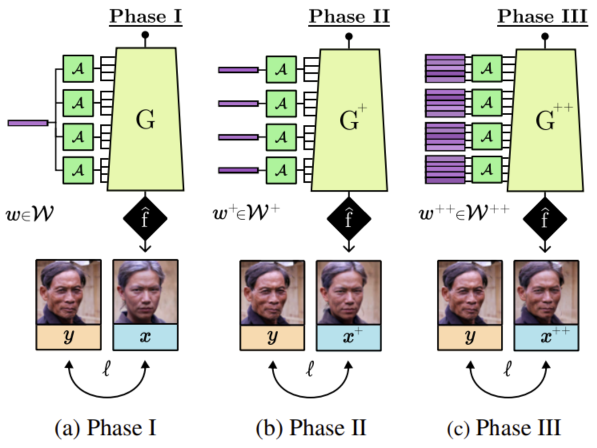


Fig. 5. Example of the three phase latent extension used in the benchmark model

In their work, they take advantage of 3 stage latent space expansion. Using a pre-trained SyleGAN model, their first stage performs global style modulation to resolve a single latent vector. Stage 2 performs layer-wise latent extension to solve for a latent matrix. Stage 3 performs filter-wise extension to solve for a final output tensor of the restored image [3]. An overview of the algorithm is given in algorithm 1.

Algorithm 1: Robust StyleGAN inversion.

```
   Output: restored image $\boldsymbol{x}^{++}$
   # Phase I
1  $\boldsymbol{w} = \mathbb{E}_{\tilde{\boldsymbol{w}} \in \mathcal{W}}[\tilde{\boldsymbol{w}}]$
2  for 1 to 150 do
3      $\boldsymbol{x} \leftarrow G(\boldsymbol{w})$
4      $\boldsymbol{w} \leftarrow 0.08\bar{\nabla}_{\boldsymbol{w}}\ell(\hat{f}(\boldsymbol{x}), \boldsymbol{y})$
   # Phase II
5  $\boldsymbol{w}^+ = \text{repeat}(\boldsymbol{w}, N_L)$
6  for 1 to 150 do
7      $\boldsymbol{x}^+ \leftarrow G^+(\boldsymbol{w}^+)$
8      $\boldsymbol{w}^+ \leftarrow 0.02\bar{\nabla}_{\boldsymbol{w}}\ell(\hat{f}(\boldsymbol{x}^+), \boldsymbol{y})$
   # Phase III
9  $\boldsymbol{w}^{++} = \text{repeat}(\boldsymbol{w}^+, N_F)$
10 for 1 to 150 do
11     $\boldsymbol{x}^{++} \leftarrow G^{++}(\boldsymbol{w}^{++})$
12     $\boldsymbol{w}^{++} \leftarrow 0.005\bar{\nabla}_{\boldsymbol{w}^{++}}\ell(\hat{f}(\boldsymbol{x}^{++}), \boldsymbol{y})$
13 return $\boldsymbol{x}^{++}$
```

The custom models were compared to the baseline models using both loss values through training as well as subjective evaluation using the smaller old-image dataset that doesn't contain ground truth labels.

D. Model Training

The entirety of the training experiments were conducted using Kaggle's jupyter notebook interface and their GPU P100 to keep up with the heavy neural network. Although this was a solid option, it did still have limitations, leading to less training than would be optimal. Due to these limitations, the first training session went for only ten epochs.

To prepare for training, the weights were initialized randomly using a Normal distribution with a mean of 0 and standard deviation of 0.02. The generator model was then defined and created using the layers described before, and provided the summary shown in figure 3 when provided a 256x256 one channel image as input. The discriminator model then went through the same process and provided the summary shown in figure 4 with 256x256 one channel image as input. The two loss functions for adversarial loss using BCEWithLogitsLoss and pixel wise loss using L1Loss. For both the generator and discriminator the Adam optimizer was used with a learning rate of 0.0001 and a beta value ranging from 0.5 to 0.999. Each epoch of the training session would run through the whole training set with a batch size of 4, which was less than ideal but was the best option to fit in the GPU memory. For each batch, the generator would first go through the training process. The generator's optimizer gradient was zeroed, and "real" and "fake" images were generated by the generator and discriminator. The loss was then calculated by adding the adversarial loss from the "fake" images and the pixel wise loss from the "real" images and the actual clean images. The loss was then used for backpropagation and the optimizer would step. The discriminator was then trained through a similar process, starting by zeroing the gradients of the discriminator optimizer. For the discriminator the "real" images and loss were predicted by the discriminator model with the clean images and the adversarial loss. The "fake" images were predicted by the discriminator using the earlier generated image, and the "fake" loss again with adversarial loss but with the "fake" predictions. The discriminator loss was defined as the average of the "real" and "fake" loss. This loss was then used for backpropagation and the optimizer would step. For each iteration of the data loader the generator and discriminator loss were recorded. For the first training session, after ten epochs the training was finished and ready to be evaluated.

E. Model Evaluation

To be able to fully evaluate the development of the model over training, a version of the model was saved after each epoch. To evaluate the model, objective and subjective evaluations were completed. For an objective evaluation, the loss was first considered. The validation set was then used to calculate the accuracy and loss of the generators from each epoch using the damaged Imagenette test set as input. The final testing would be done as a subjective evaluation first using the Imagenette validation set, and then using the daguerreotypes dataset.

III. Results

A. *Objective Evaluation on the Custom Models*

The objective evaluation consisted of analyzing the accuracy and loss values of the generator and discriminator. During each training epoch, the model was saved and the loss of the best model was kept. Below shows the loss values versus epoch of the generator in the GAN.

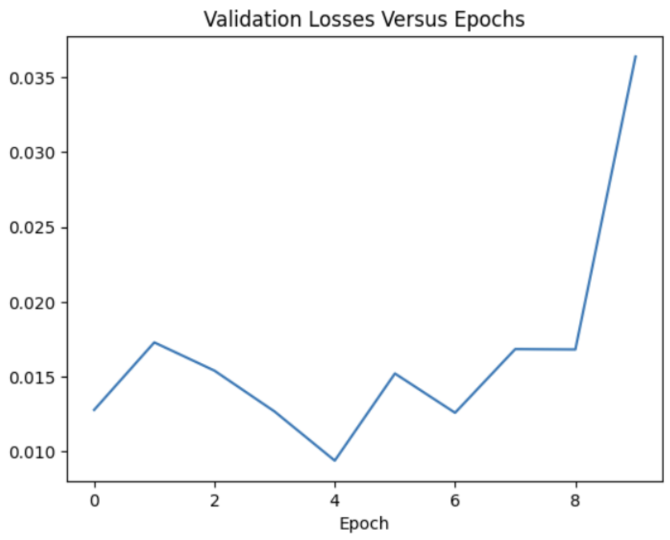


Fig. 6. Validation loss versus epochs for the custom GAN generator

It can be seen that at epoch 4, the model is at its lowest loss. The model from the 4th epoch was saved and used for future sections.

B. *Subjective Evaluation on the Custom GAN*

Below shows an example of the output of the GAN with an input image from the modified imagenette dataset and the ground truth image in figure 7.

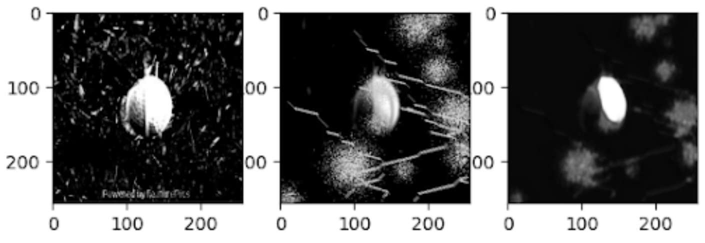


Fig. 7. clean image (left), damaged image (center), restored image (right)

It can be seen that the model does not quite repair the image as intended in this case. The model removes the cracks and minimizes the splotches, but in doing so it makes the whole image blurrier. It does not sharpen the image or recover any of the lost intricacies, but instead does the reverse.

Additionally, below shows the output from a sample in the subjective evaluation dataset of daguerrotypes in figure 8.

It can be seen again that the DCGAN does not perform as intended, making the image quite blury.

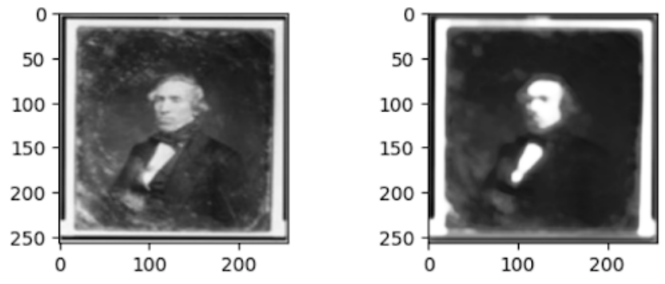


Fig. 8. naturally damaged image (left), restored image (right)

C. *Subjective Evaluation on StyleGAN*

Unfortunately, the code provided by the researchers was faulty and could not be run.

IV. Conclusion

The goal of this project was to develop a custom DCGAN to repair damaged historical images that have been subject to a variety of damages.

As can be seen from the subjective evaluation though, the model did not perform quite as anticipated, making the images blurrier. One likely reason for this, and for many other GANs, is that the loss value is quite abstracted from a semblance of quality or desired results.

Additionally, limited GPU resources limited the amount of training and hyperparameter tuning that could have been done.

External Sources Used

For this project, several external sources were used in the coding phase. To create the custom DCGAN, the Pytorch article ”DCGAN Tutorial” was used [4]. In addition, Chat GPT was used to help develop the code used in the training loop.

Contributions

Martin Trpceski: Designing custom model and finding the baseline model, creating code to train models, training models, working on paper, and working on slides.

Sean Gleason: Created image processing code to create the dataset, trained models, evaluated the benchmark model, worked on paper, and worked on slides.

References

[1] J. Howard, et al, ”imagenette”, 2022. https://github.com/fastai/imagenette

[2] ”Daguerreotypes” Library of Congress, Prints & Photographs Division, [reproduction number, e.g., LC-USZ62-12345]

[3] Zhendong Wang, undefined., et al, ”Diffusion-GAN: Training GANs with Diffusion,” 2023.

[4] Y. Poirier-Ginter, J. Lalonde, ”Robust Unsupervised StyleGAN Image Restoration,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023, pp. 22292-22301.

[5] N. Inkawhich, “DCGAN Tutorial,” Pytorch. https://docs.pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html (accessed May 08, 2025).