Generative Adversarial Denoising Autoencoder for Face Completion

Avery Allen, Wenchen Li

Project Overview

Our original project focus was creating a pipeline for photo restoration of portrait images. Many damaged photos are available online, and current photo restoration solutions either provide unsatisfactory results, or require an advanced understanding of image editing software. Shortly after beginning the project, we narrowed our scope to what we consider the most interesting subproblem of photo restoration: facial image completion. The image completion problem we attempted to solve was as follows, given an image of a face with a rectangular section of the image set to be white, fill in the missing pixels.

Original image and masked image.

There are previous works that attempt to solve similar problems. **Graph Laplace for Occluded Face Completion and Recognition** and **Partially occluded face completion and recognition** both leverage a large image database to find similar faces to use to complete the missing patch, but results are only shown for low resolution grey scale images. **Image Denoising and Inpainting with Deep Neural Networks** uses deep networks pre-trained with denoising autoencoders for image inpainting, but do no show completion of large missing patches.

Our approach builds upon the work presented in **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks** and **Enhancing Images Using Deep Convolutional Generative Adversarial Networks (DCGANs)**. In our approach we use a generator, a collection of convolution and deconvolution layers, to reconstruct the original unmasked image. The generator takes the form of a fully convolutional autoencoder. A discriminator is also trained using the output of the generator. The discriminator's output is used together with a reconstruction cost to update the weights of the generator.

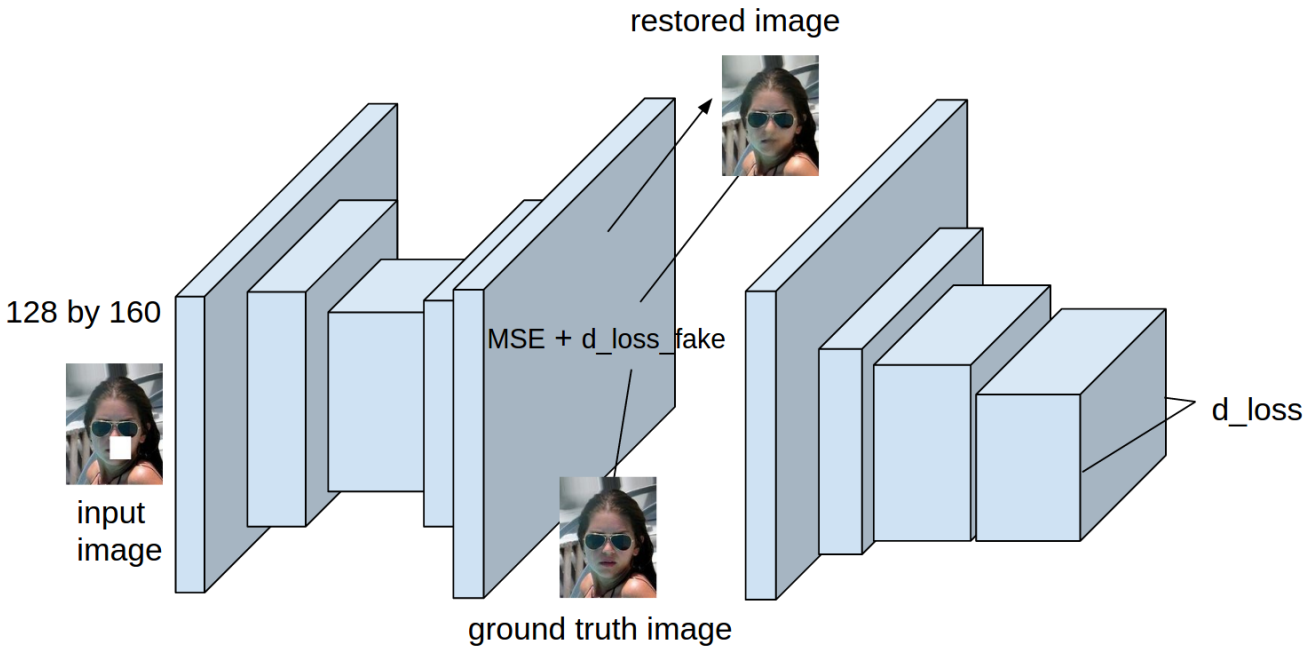
Dataset

We use the **CelebA Dataset** to train our model. The dataset consists of around 200,000 images including over 10,000 unique identities. We reserve 1000 images as a test set, and 1000 images as a validation set. For each image, we set a randomly sized patch to be white. Each generated patch is at least 10 pixels in width/height and takes up a maximum of 1/3 of the image. The patch's X,Y coordinates are selected randomly.

Architecture

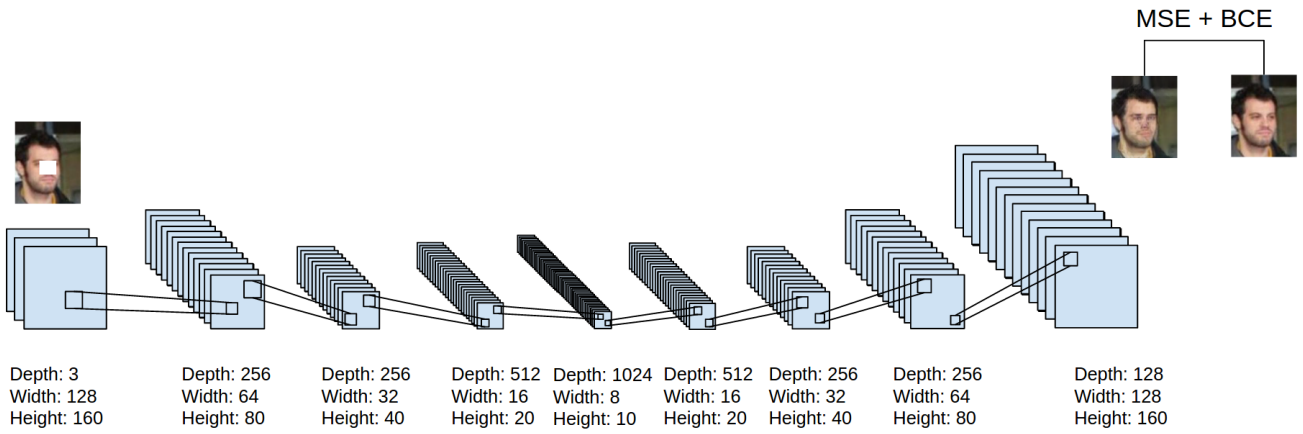
Overview

The architecture has two major components: the autoencoder, and the discriminator. A diagram of the architecture is shown below. The autoencoder (left side of diagram) accepts a masked image as an input, and attempts to reconstruct the original unmasked image. The discriminator (right side) is trained to determine whether a given image is a face. The discriminator is run using the output of the autoencoder. The result is used to influence the cost function used to update the autoencoder's weights. We built upon **Enhancing Images Using Deep Convolutional Generative Adversarial Networks (DCGANs)**'s codebase. The project code can be found in this **repository**.



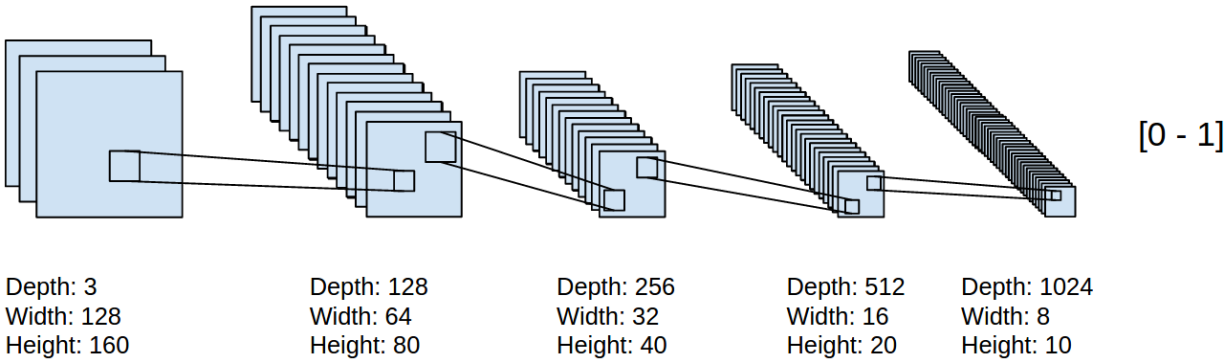
Autoencoder

Traditionally, autoencoders have been used to learn a feature representation for some data set. Denoising autoencoders artificially corrupt input data in order to force a more robust representation to be learned. In our case, the image mask is the data corruption. Our autoencoder first transforms the input data through a series of 4 convolution layers. At the center layer of the autoencoder, the image is represented by 1024 8 x 10 feature maps. The input image is then reconstructed by 4 fractionally strided convolution layers (sometimes called deconvolution layers). The output layer uses a tanh activation function. The autoencoder's weights are updated by a linear combination of two costs: the mean squared error between the original unmasked image and the reconstructed image, and binary cross entropy using the output of the discriminator (described in the Loss Functions section). A diagram of the autoencoder is shown below.



Discriminator

The discriminator consists of 4 convolutional layers. It accepts a 128x160 RGB image as input. The discriminator is trained to determine whether the input image is a real face. A sigmoid function is used on the final layer to yield a probability between 0 and 1. The probability represents the discriminator's belief that the input image is a real face. During training, the discriminator is updated using real images from the training set, and images generated from the autoencoder. The discriminator is rewarded for assigning a low probability to generated faces, and a high probability to real faces. The cost functions are described in the Loss Functions section. A diagram of the discriminator is shown below.



DCGAN Guidelines
The architecture adheres to the following guidelines suggested in **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks** which seemed to help make training stable:

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Loss functions

There are four loss values we define in our model:

```
d_cost_real = binary_cross_entropy(p_real, T.ones(p_real.shape)).mean()
d_cost_gen = binary_cross_entropy(p_gen, T.zeros(p_gen.shape)).mean()

g_cost_d = binary_cross_entropy(p_gen, T.ones(p_gen.shape)).mean()
enc_cost = mse(source_flat, target_flat).mean()
```

The autoencoder's ability to fool the discriminator is determined by the discriminator's response when given a generated face (how well p_gen approximates p_real, g_cost_d). P_gen and p_real are both probability distributions from the sigmoid function at the last layer of the discriminator. The autoencoder is also forced to learn the pixelwise representation of the faces by learning from the mean squared error from the groundtruth images (enc_cost).

The discriminator's performance is measured by how well it can identify faces from groundtruth images (d_cost_real), and also how well it can recognize faces generated from the autoencoder (d_cost_gen).

The cost function used to update the autoencoder is:

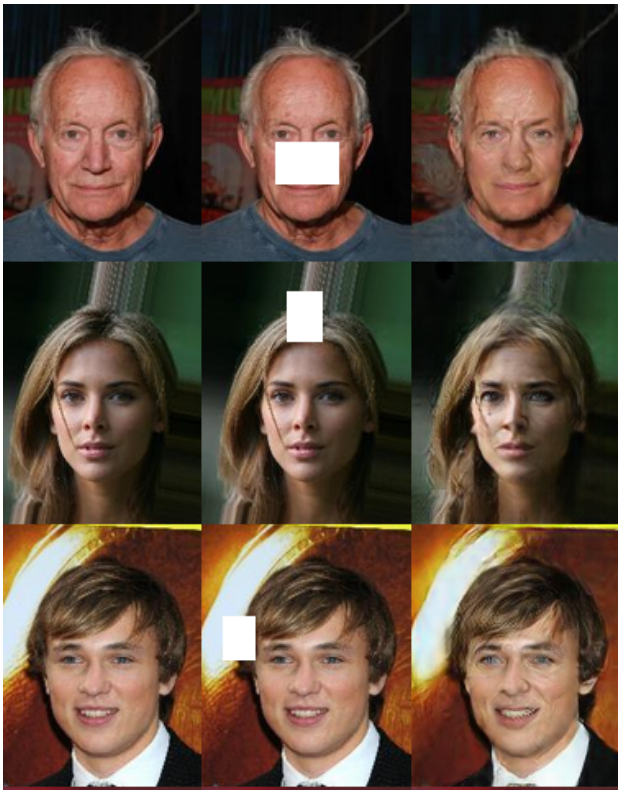
```
g_cost_d + enc_cost * X
```

X is a parameter that decides how heavily to weight MSE when calculating the autoencoder cost. The cost function used to update the discriminator is:

```
d_cost_real + d_cost_gen
```

During training and testing, we track: g_cost_d, enc_cost and d_cost_real.

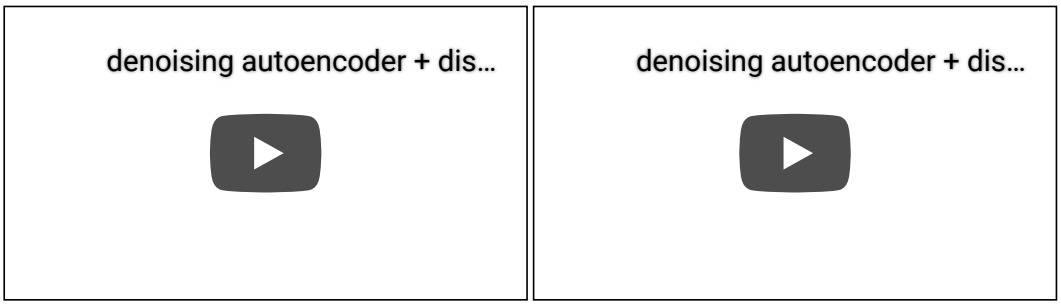
Results



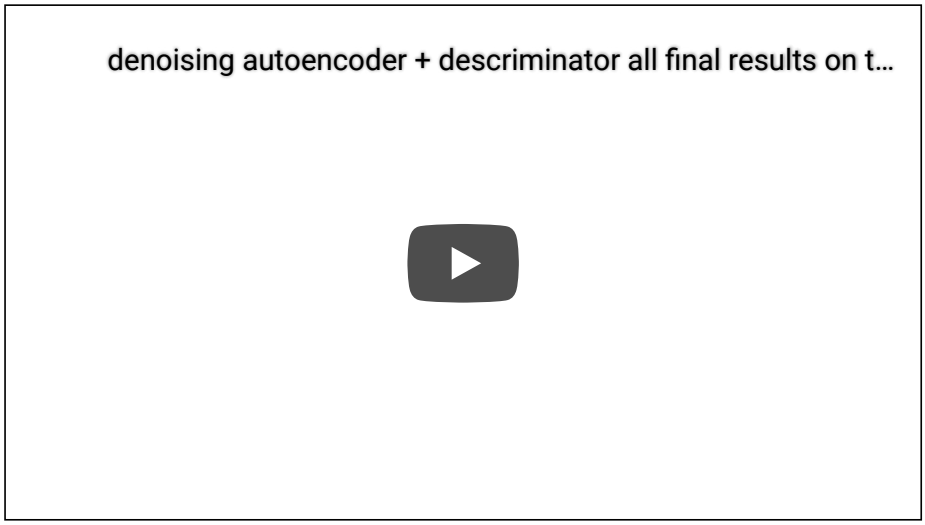
The base setup is the denoising autoencoder and discriminator with x = 1.

Base Setup Testing Set Results:
Results from images in the 1000 image testing set.

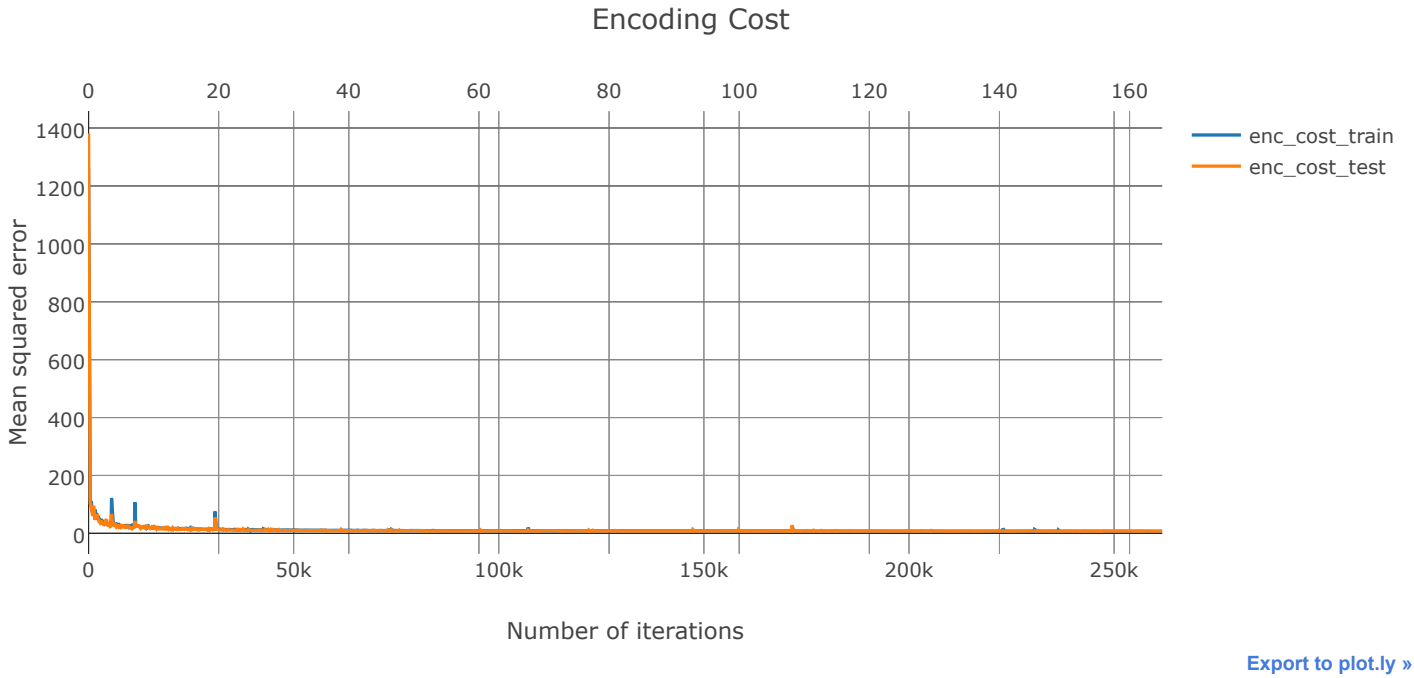
Test results along training epochs



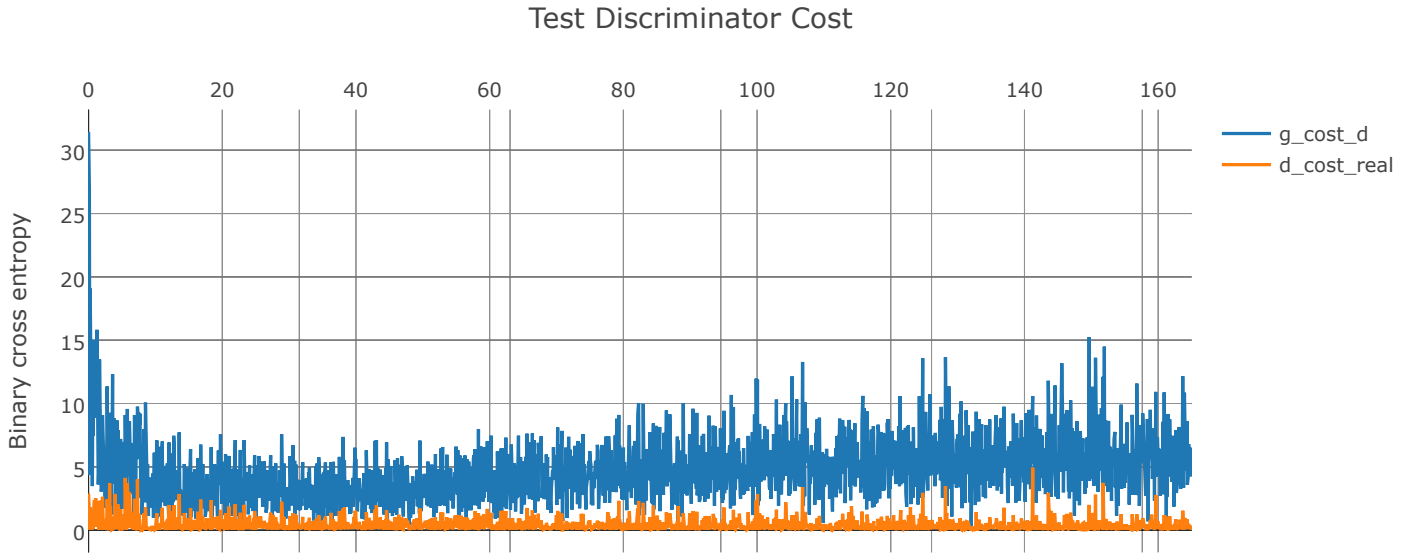
final test results

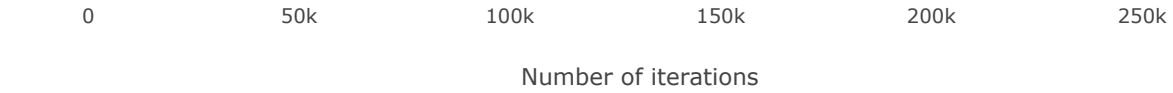


loss values along epoches



Export to plot.ly »





Export to plot.ly »

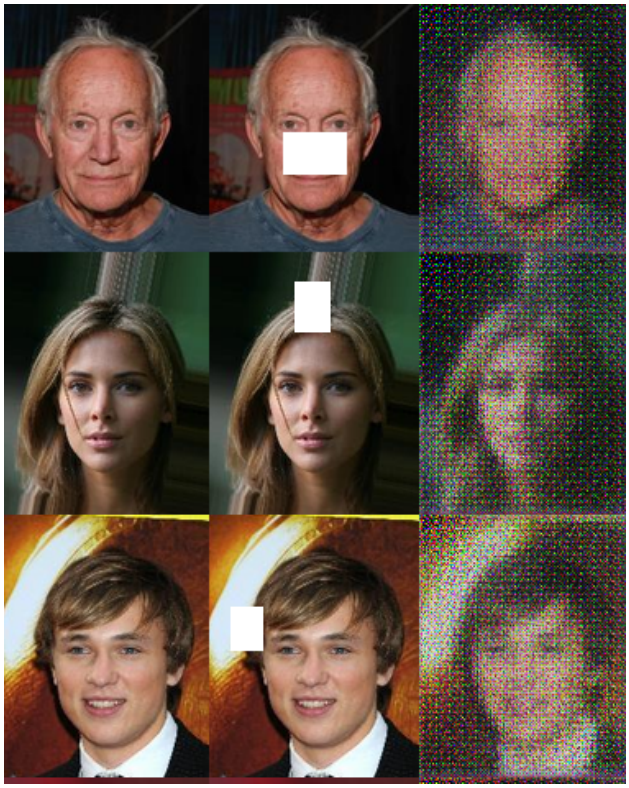
Around 10k iterations, the encoding cost reaches a local minimum. We see a divergence trend starting at around 15k between the autoencoder's ability to fool the discriminator (g_cost_d) and the discriminator's ability of identifying real face data(d_cost_real).

A possible explanation is that as the MSE encoding cost converges on a lower bound, the autoencoder is no longer able to fool the discriminator by updating with a loss function that so heavily favors the already converged MSE. A possible solution would be to use a dynamic learning mechanism to adjust the cost function for the autoencoder. As training progresses, the parameter X should be updated to weight MSE loss less once MSE cost has converged because learning from MSE no longer seems to be "efficient" in the sense of fooling the discriminator.

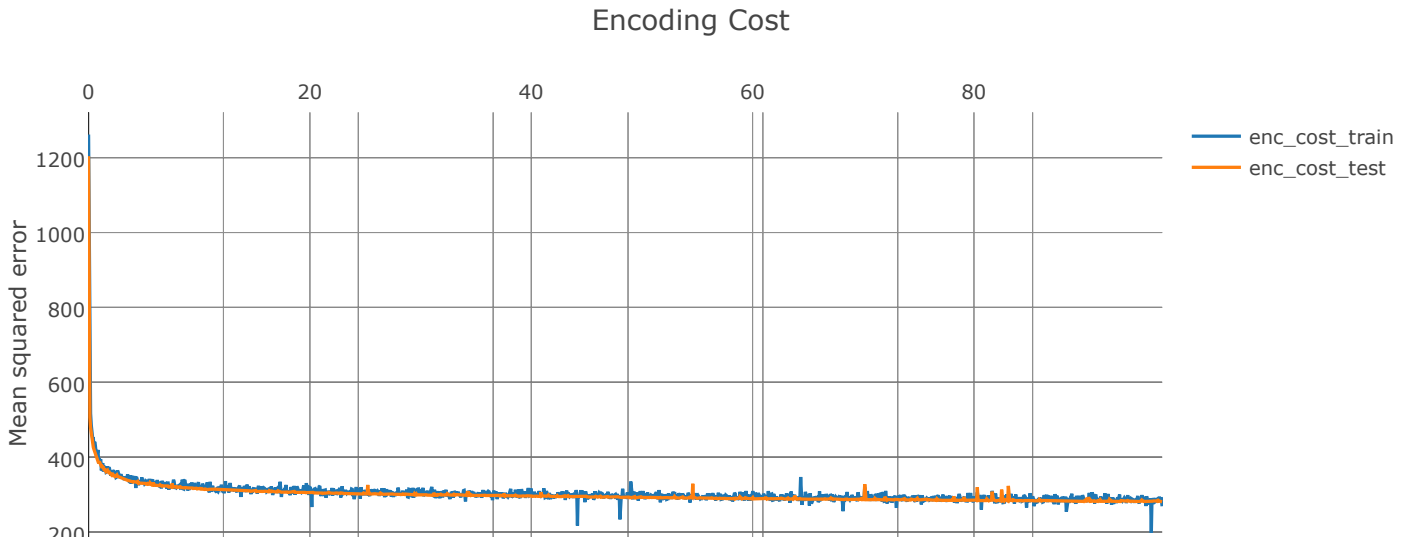
Experiments

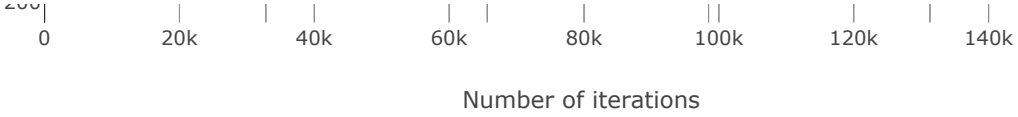
To help better understand how the autoencoder and discriminator work together, we ran several control experiments based on our initial setup.

Without Discriminator



without discriminator mse loss

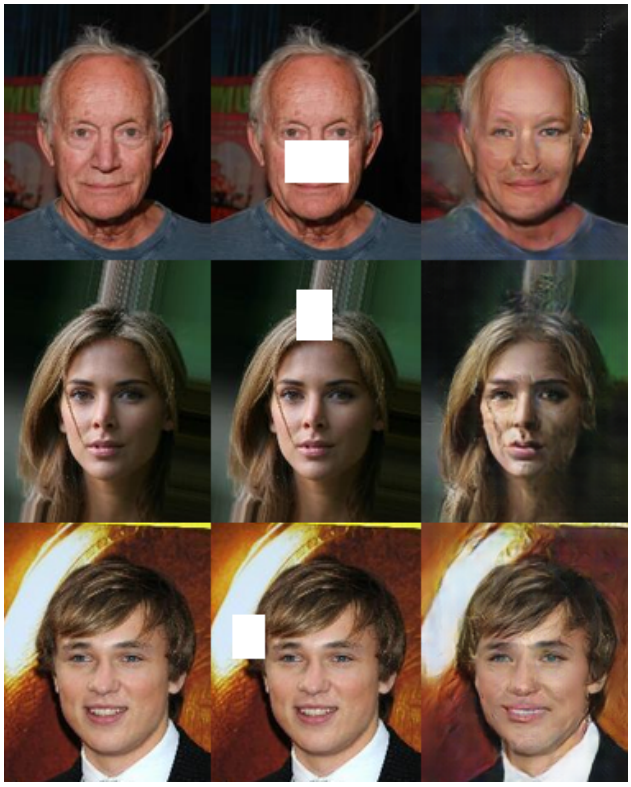


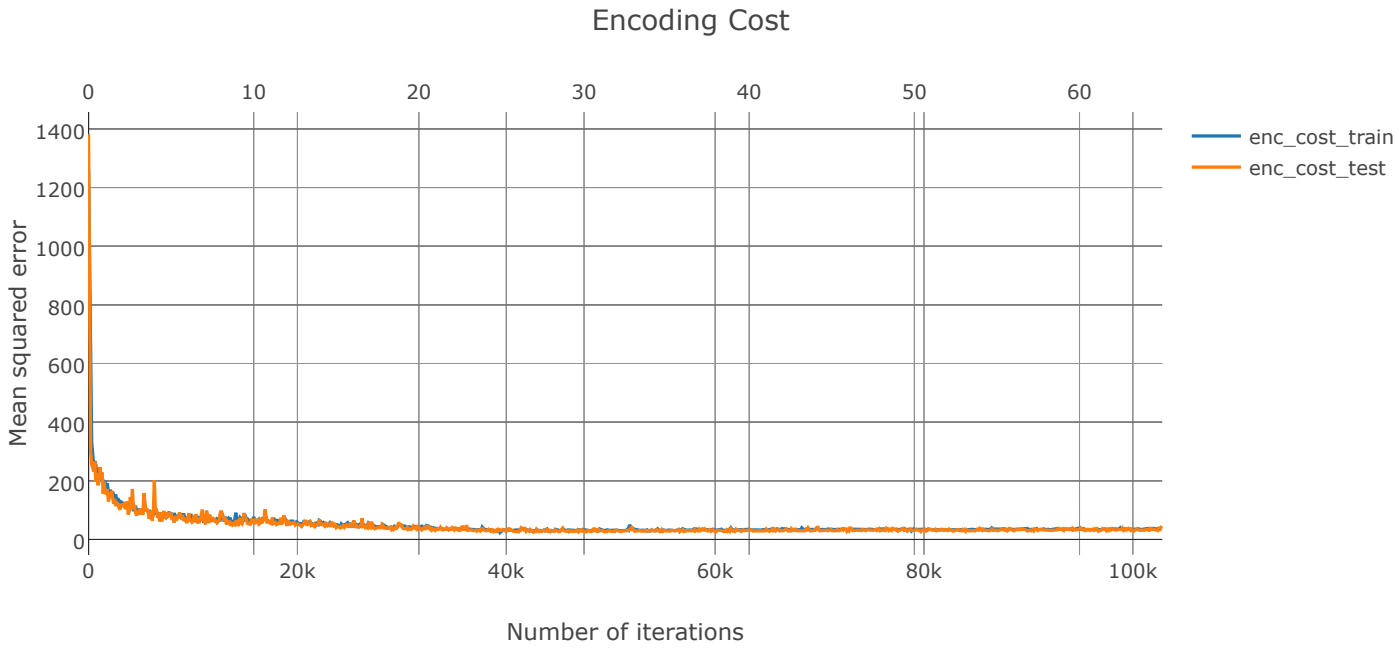


Export to plot.ly »

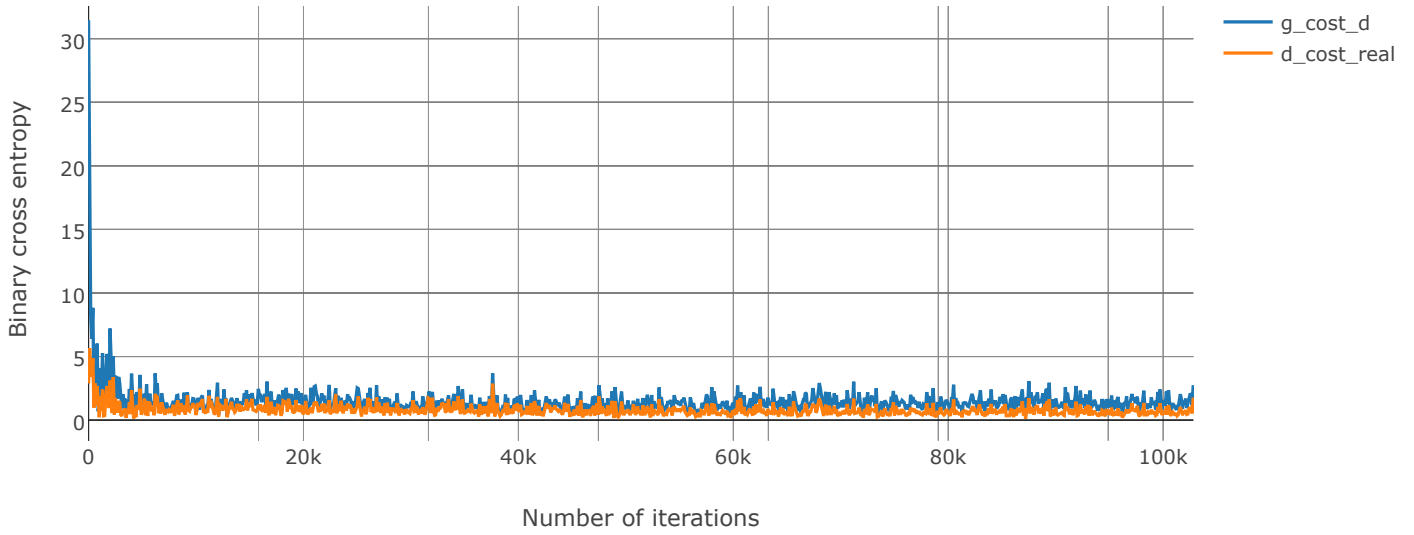
We see that the discriminator does help with reconstruction even though it's objective is to discriminate between real face data and generated face data.

Decrease Mean Squared Error's Influence on Reconstruction





Export to plot.ly »



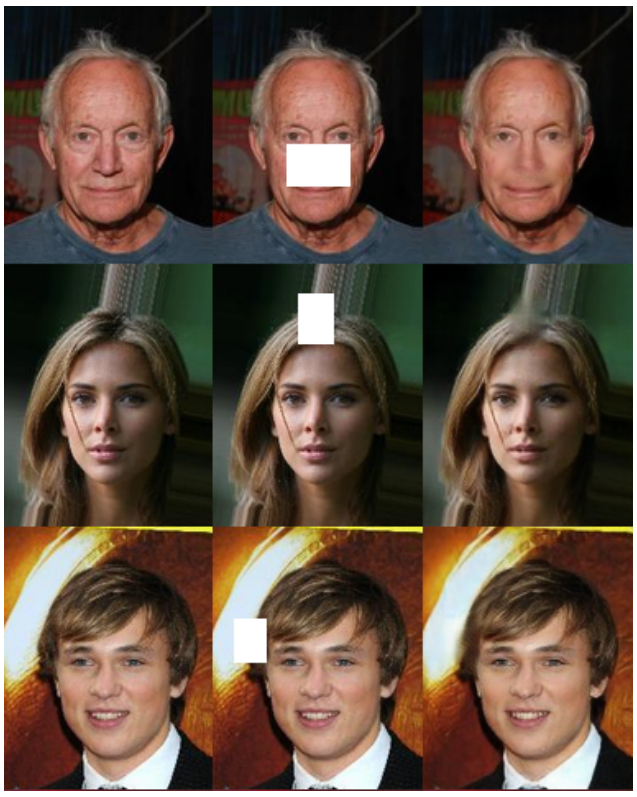
Export to plot.ly »

To experiment with how to combine MSE loss and discriminator loss for autoencoder updates, we set

```
generator_loss = MSE * X + g_cost_d
```

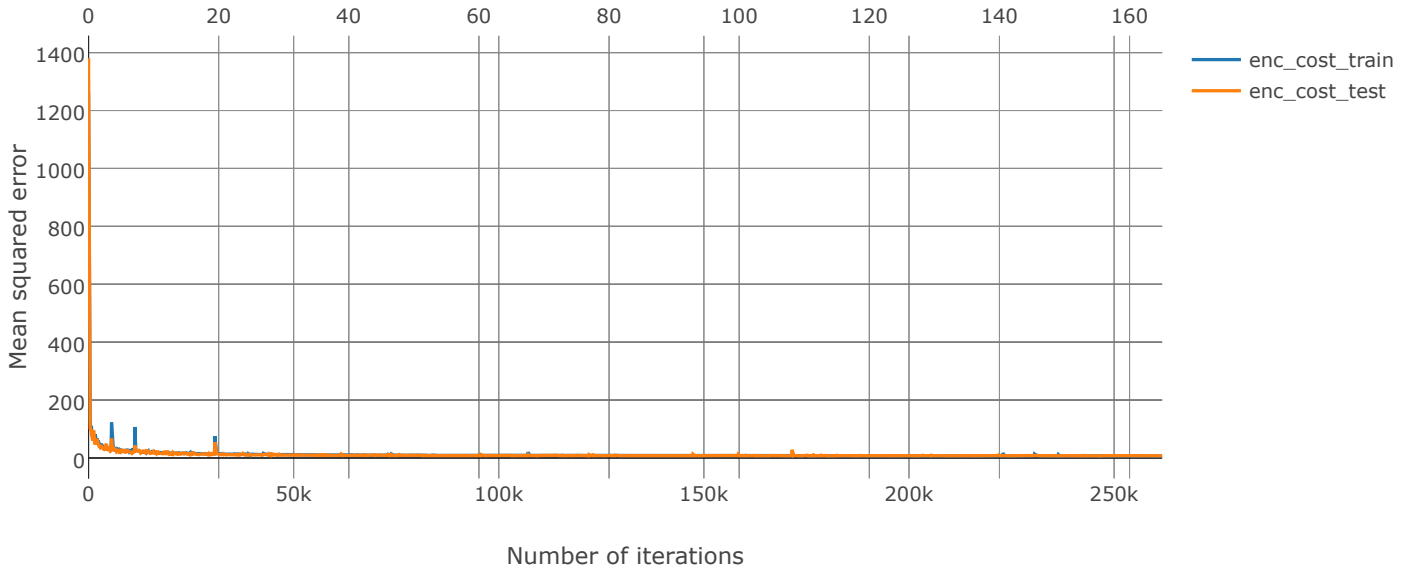
where X = .01 (base model X = 1). From the X = .01 experiment, we see reconstruction loss reach a local minimum at a loss value much higher than X = 1. This leads the generator to learn more from the discriminator and reach an equilibrium quickly, but leads to slightly worse reconstruction of the original input. We believe that generated and real image discriminator loss reaching the equilibrium point too early leads the discriminator and autoencoder to stop learning from one another. This is because both sides already believe they are doing a good job based on their objective. As mentioned before, we believe that adding a way to actively adjust the X parameter may lead to more stable training.

MSE Without Pooling



MSE Without Pooling

Encoding Cost



Export to plot.ly »

The original implementation provided by **Enhancing Images Using Deep Convolutional Generative Adversarial Networks (DCGANs)** performs the additional step of max pooling on the reconstructed and target image before calculating the mean squared error. Our base implementation also uses this method. The reason described for including max pooling is that it counteracts the blurred look that is often present when using denoising autoencoders. We trained an additional model to test whether pooling was effectively preventing blur. The results above clearly show that removing pooling does in fact bring back a blurred look to the problem area. However, the areas of the image that were not masked in the input image seem to be mostly clear of artifacts. This is in direct contrast from when pooling is used; images often have artifacts on areas outside of the original image mask.

Future Work

- From the experiments, we see a dynamic mechanism is necessary for the autoencoder to learn more effectively. Adjusting the weight between MSE loss and discriminator loss on the fly to avoid divergence could be advantageous. We want to explore more in this area.
- After training is complete, we only use the autoencoder portion of the architecture. However, we have also trained a discriminator that can identify faces. We'd like to measure how well the discriminator performs in a face recognition pipeline.