DCGAN Image Completion

Deep Learning Final Project

Shay Malkin, Dan Levy

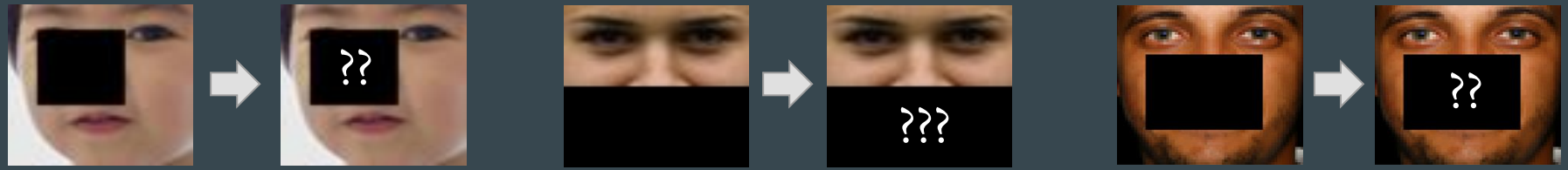
Lecturer: Dr. Teddy Lazebnik

Holon Institute of Technology, Faculty of Sciences,

Applied Mathematics Department, 2020

Introduction - The Problem

Given a masked image, meaning an image with a nullified area, we would like to train a neural network model that will generate an appropriate part to complete the image with in a way that looks natural to the human eye.



This is a non trivial problem. In order to solve it, we need to be able to generate the missing part of the image from scratch so that it both looks real and is compatible to the rest of the image. In this project, we used a Generative Adversarial Network to create those images.

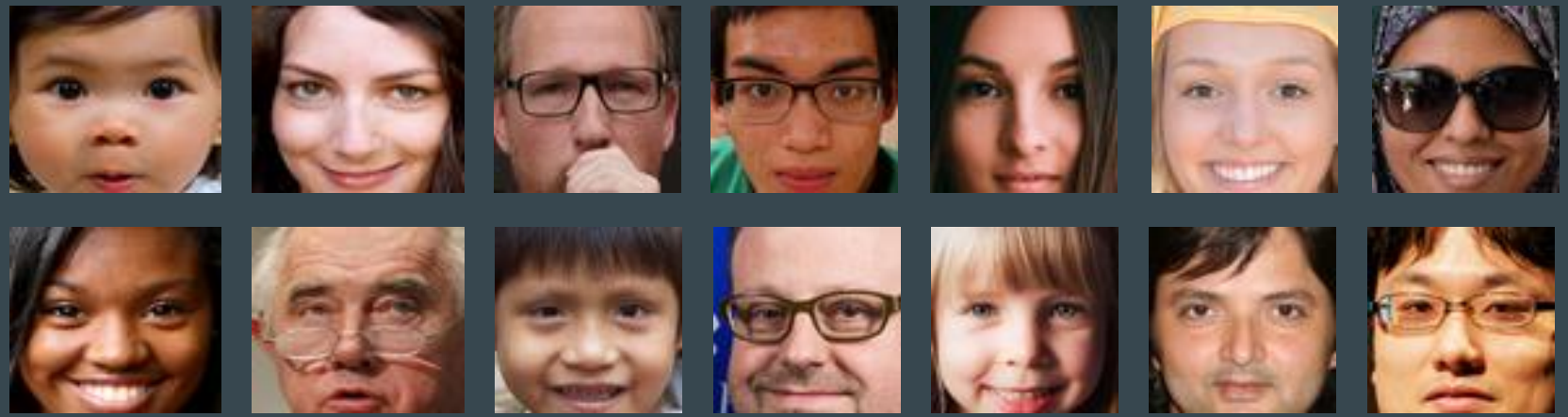
Motivation

We chose this project because we find Generative Adversarial Models fascinating. They are a clever way of using Deep learning techniques to solve unsupervised learning tasks. Generally speaking, a generative model is able to learn to approximate the distribution of a given data set and generate new samples from that distribution.

We chose to tackle this particular problem of completing images because it represents a fun, visual way of using generative models. For a computer to be able to generate images that look authentic is pure magic! (not really... it's actually math..)

The Data Set

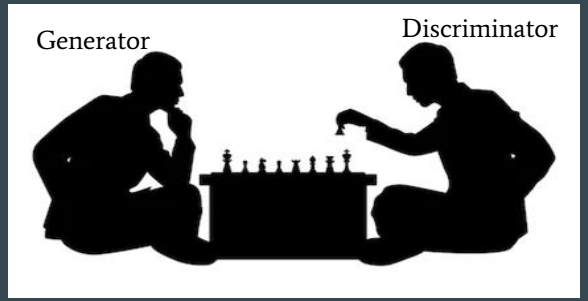
The data set used for this project is the FFHQ Faces data set. It contains 70,000 images of size 128 x 128 x 3 of human faces. The images were cropped to a size of 64 x 64 x 3 and 30,000 images in total were used for training the model.



DCGAN: General View

DCGAN stands for Deep Convolutional Generative Adversarial Network.

It consists of two components - a generator and a discriminator, of which each one's "goal" - minimizing its loss function, contradicts the competing model's success. These two components train together simultaneously, playing this competitive game, where each one tries to "defeat" the other, making them both better at their respective tasks as the game goes on.. As one improves, it forces the other one to improve as well and so on and so on...



Discriminator - general idea

The discriminator, D(Y), is a classifier, that gets an image Y as input and returns a scalar between 0 and 1, that is the probability of the input being a “natural” image.

The output will approach to 0 when the image is likely to be from an unknown distribution, meaning artificially generated by G(Z).

Values closer to 1 will indicate that the image belongs to the data distribution, which means in our case a human face image “naturally” captured by a camera.

Discriminator - architecture and features

Network architecture: CNN Binary Classifier

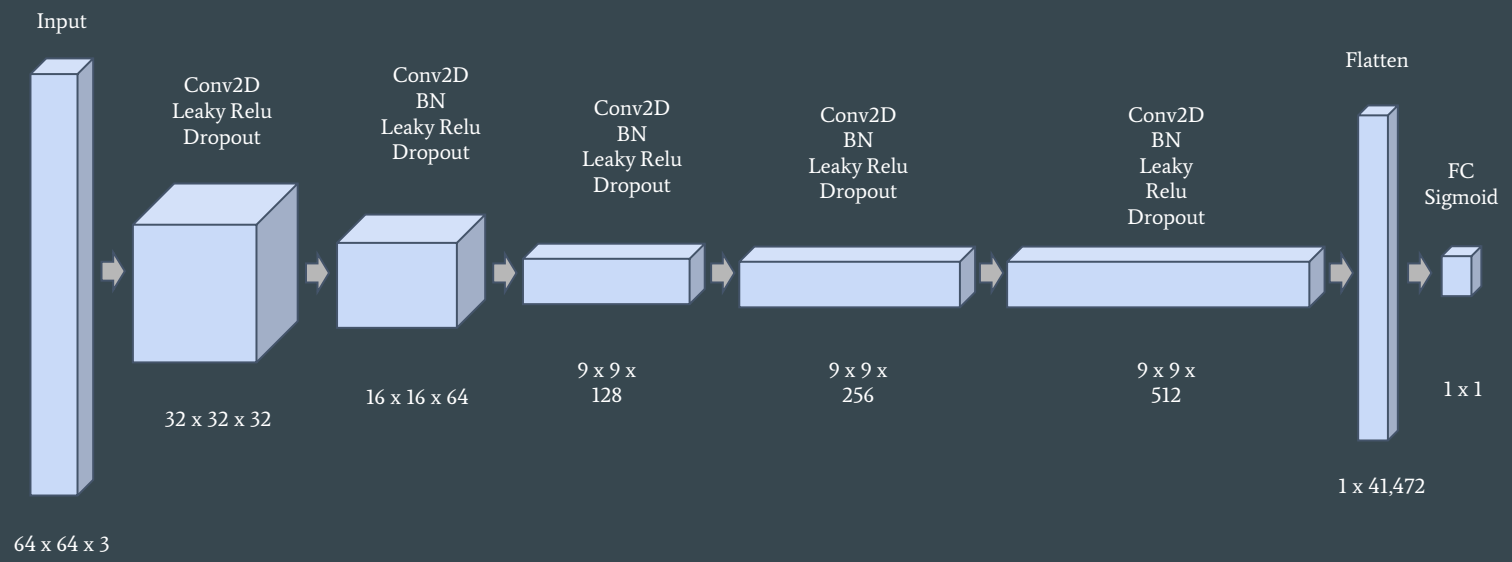
Output activation function: sigmoid

Loss function: binary cross entropy

Learning type: supervised

The Discriminator alternately gets images generated by the the generator with label 0 and images taken from the data set labeled by 1. It ‘s job is to learn to distinguish between the real images and the fake ones.

Discriminator
Architecture



The architecture was designed following the guidelines from the DCGAN Paper by Alec Radford and Luke Metz

Generator - general idea

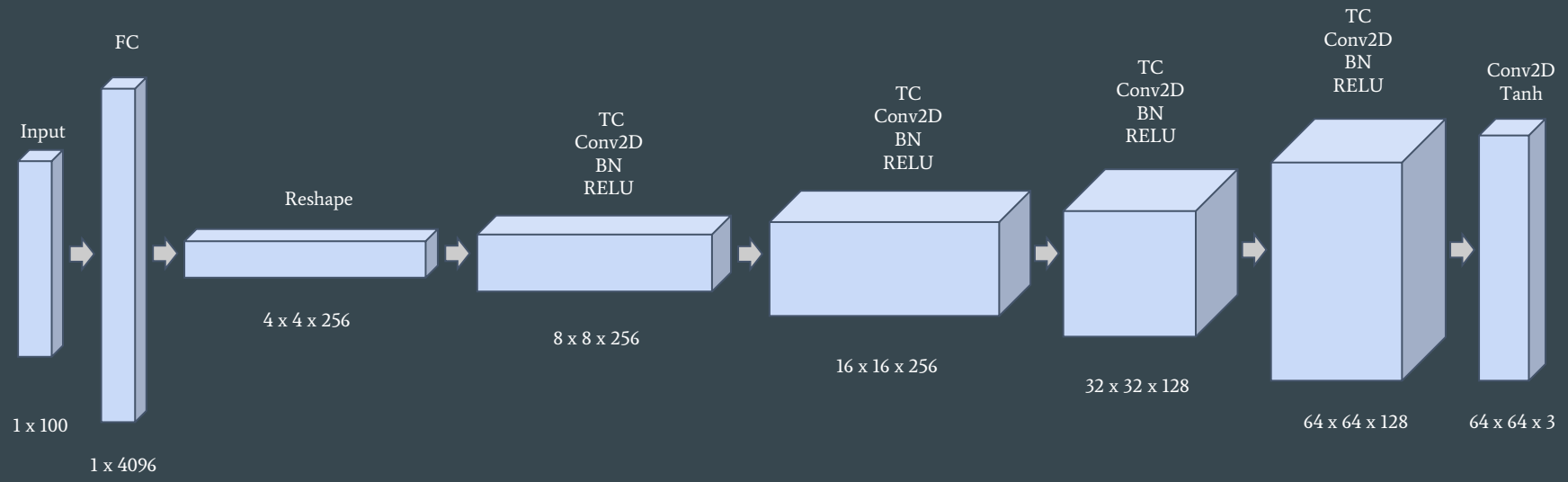
The Generator, G(Z), gets a vector Z sampled from a known distribution and returns an image Y. The input vector Z is of size 1 x 100 and is generated from a normal distribution with mean 0 and variance 1. The output tensor is an image with the same dimensions as the images from the data set (64 x 64 x 3).

The Generator up-samples the input gradually through it's layers, using a transpose convolution, until the desired dimensions are obtained.

The Generator is trained to "fool" the discriminator by learning to produce images that will be classified as if they belonged to the data's known distribution.

Generator

Architecture



The architecture was designed following the guidelines from the DCGAN Paper by Alec Radford and Luke Metz

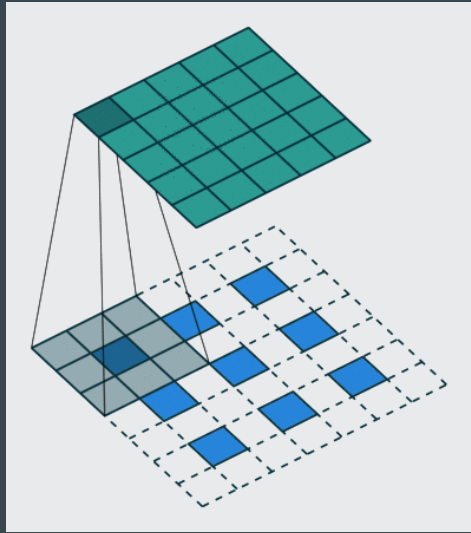
Generator - general idea

Unlike many other convolutional neural network which perform downsampling of images, the generator is composed of transpose convolution layers that upsample the input vector into an image that is a tensor of the dezired dimensions.

Transpose convolution intuition

As we know, a standard convolution extracts high level features from an image (or lower level features).

A transpose convolution does the opposite. It takes an input of high level features and extracts the image (or lower level features) that they represent.



The training process

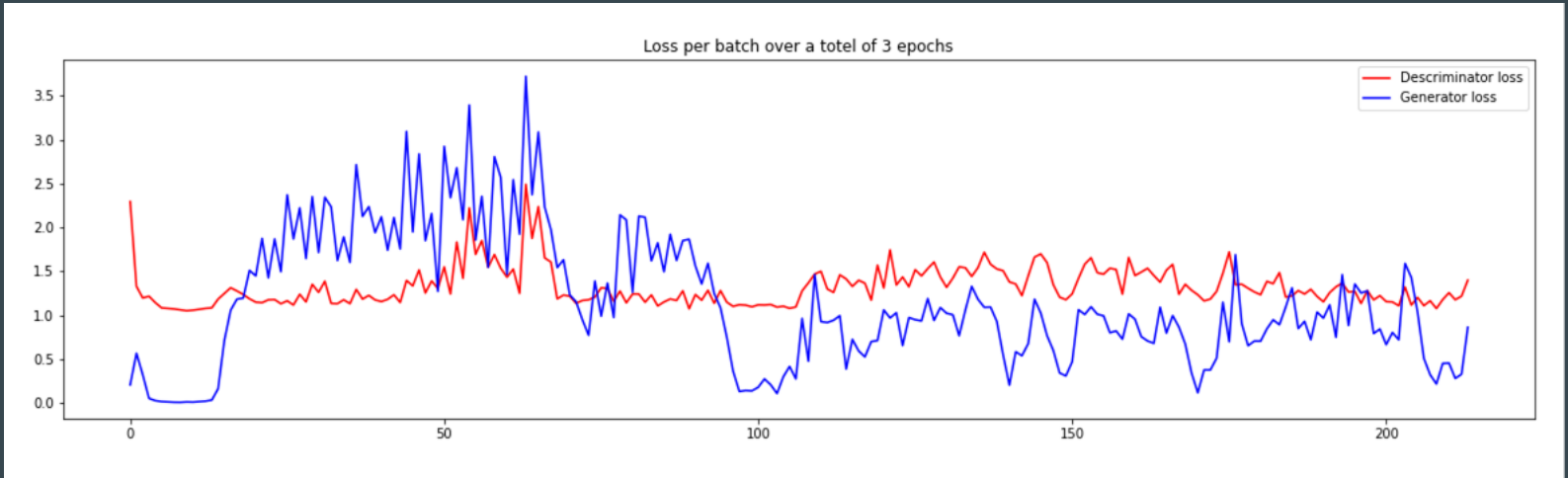
$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

For every batch,

- We sample a half batch of real images taken from the data set and a half of generated images, label them with 1 and 0 respectively.
- We then train the discriminator over this batch, maximizing the expression above with respect to the discriminator (D).
- After updating the discriminator, we update the generator, minimizing the expression above with respect to the generator (G)

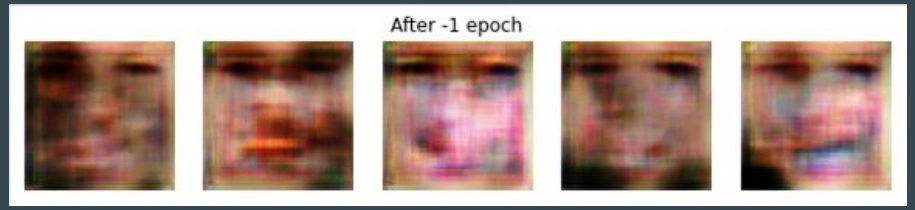
The training process

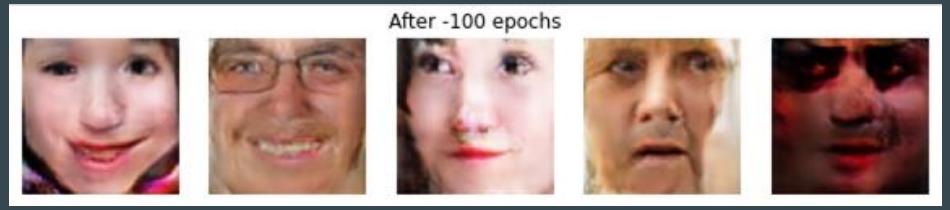
In this context, the score of each component is meaningful only with respect to the other's score. The graph below shows the "back and forth" training process between the Generator and the Discriminator.

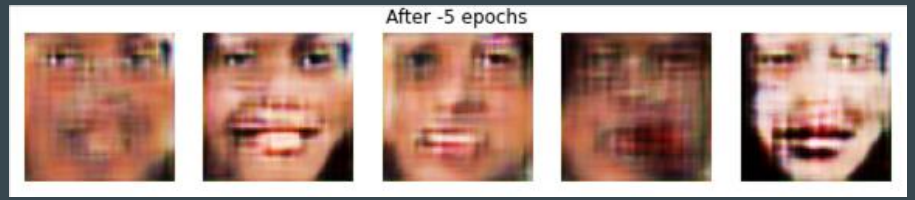


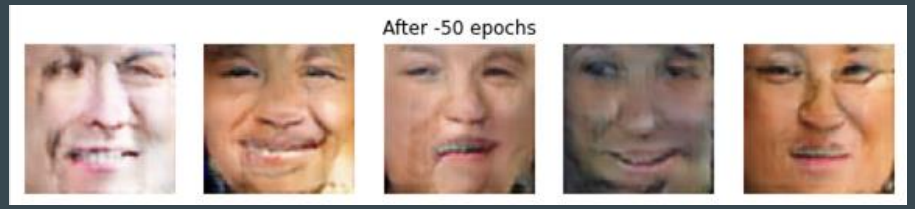
The training process

Here are the images produced by the Generator during training. We can see that the results improve as the training progresses. In the end, the images are quite impressive!









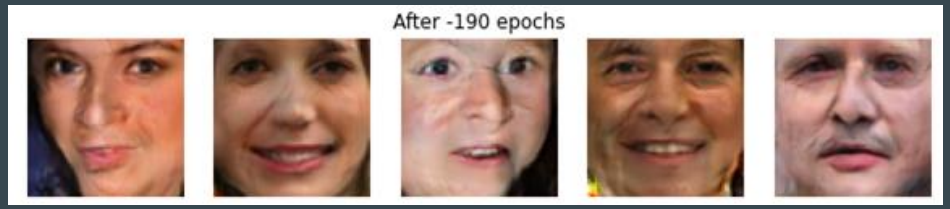


Image completion

Having now an incomplete image on one hand, and a trained generator that produces a natural looking image on the other hand, we would like to generate an image that most resembles the one we wish to complete.

For that aim, we'll be optimizing the following loss function with respect to the input vector Z, searching for $\hat{z} = \arg\min_z \mathcal{L}(z)$, so that $G(\hat{z})$ will be the most compatible image. Then we'll be able to fill the missing part with the corresponding pixels.

$$\mathcal{L}(z) \equiv \mathcal{L}_{\text{contextual}}(z) + \lambda \mathcal{L}_{\text{perceptual}}(z)$$

Image completion

The loss function is composed of 2 terms, perceptual loss and contextual loss.

$$\mathcal{L}(z) = \mathcal{L}_{\text{contextual}}(z) + \lambda \mathcal{L}_{\text{perceptual}}(z)$$

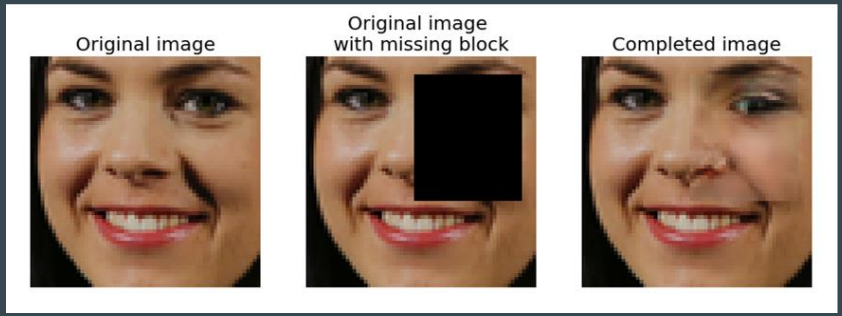
The perceptual loss measures how “real” the generated image looks:

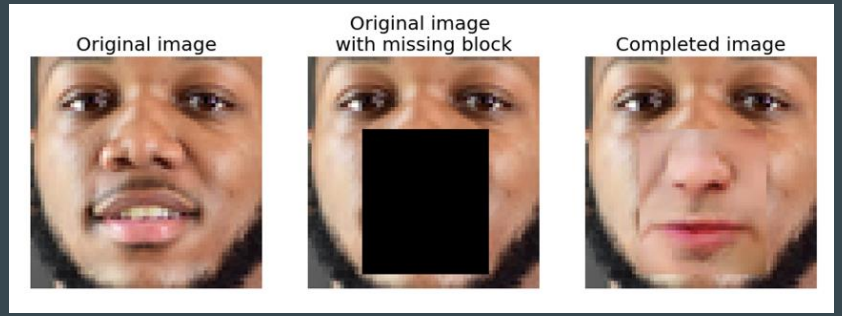
$$\mathcal{L}_{\text{perceptual}}(z) = \log(1 - D(G(z)))$$

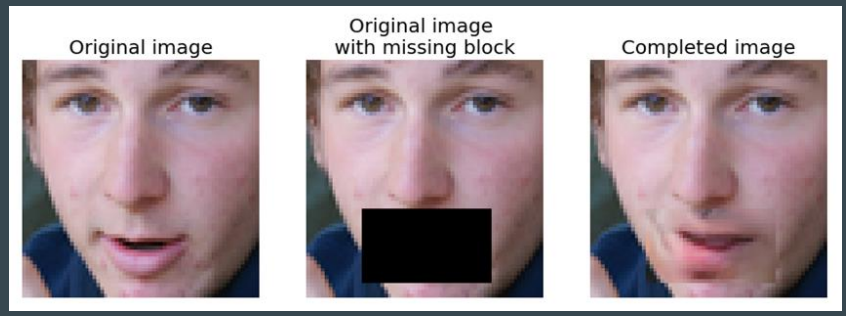
The contextual loss describes the similarity between the real and the generated image by measuring the L1 distance between the pixels around the missing part in the real image and the corresponding pixels in the generated one:

$$\mathcal{L}_{\text{contextual}}(z) = ||M \odot G(z) - M \odot y||_1$$

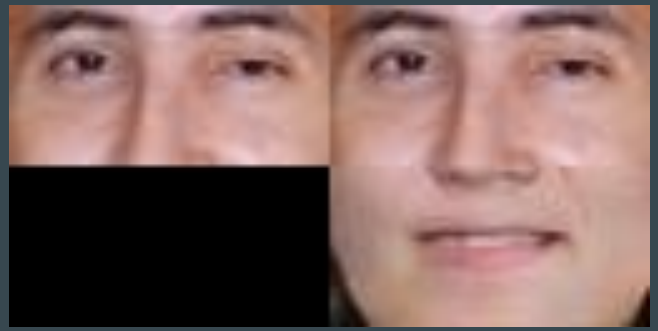
Final Results

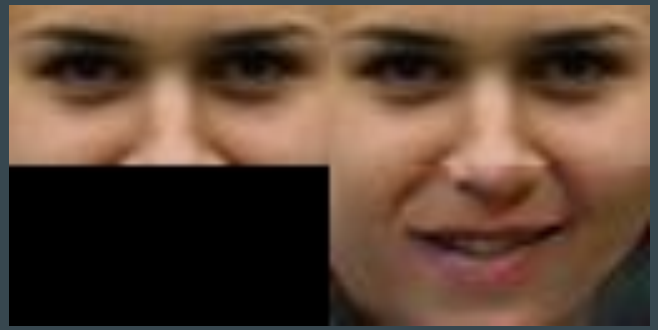






More Results





References

- [1] Alec Radford & Leka Metz, Indico Research, Boston, MA, Soumith Chintala, Facebook AI Research, New York, NY, "UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS" (2016)

- [2] Ian J. Goodfellow, Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair† , Aaron Courville, Yoshua Bengio, Department of Computer Science and Operational Research, University of Montreal, Montreal, "GENERATIVE ADVERSARIAL NETS", (2014)

Understanding the Generator

The Generator is a function that maps each vector from the latent space Z to an image. We can further investigate this latent space and what it represents by choosing two points at random and connecting them with a straight line. We then generate images from evenly spaced point along that line. The result is a smooth transition from on image to another. This implies that the latent space is continuous (loosely speaking).