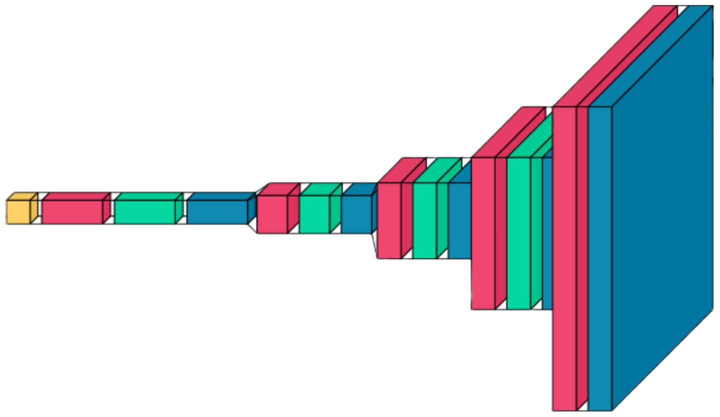
Generative Learning Via GAN

Model Architecture

Generator:

Overview: the generator task is to generate new data instances. In our task, these new data instances are real-life celebrity faces with the help of both discrete and continuous noise vectors that we generated. The model's input will be the noise vectors we created earlier, and the output is a fake image.



Input: Takes in a noise vector from a latent space of dimension latent_dim=128 and a binary attribute vector with num_attributes=40.

Architecture

Latent Space Input (Yellow): The generator begins with an input from a latent space of dimension latent_dim=128. This input is typically a 1D tensor (often visualized as a line or small rectangle) of random numbers, which acts as a seed for generating data.

Attribute Injection (Red): After the latent space, the architecture shows an injection of attributes, which correspond to specific features that we want the generated data to exhibit (attributes like "blonde," "wearing glasses," etc.). These attributes are concatenated with the latent vector.

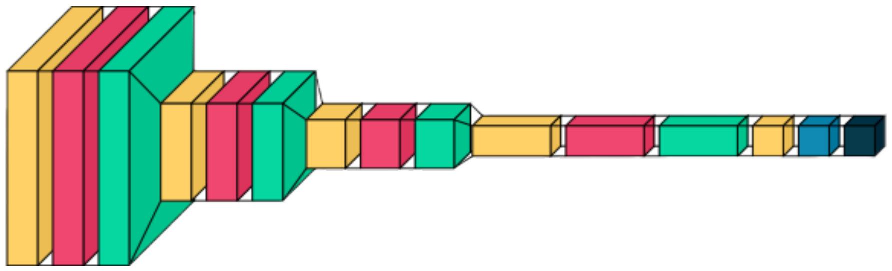
Initial Transformation (Green): The combined tensor is then passed through an initial transformation, which is often a fully connected layer or a transposed convolution (also known as a deconvolution), that reshapes it into a 3D tensor suitable for convolutional operations. And after each transposed convolution, except for the final output layer. Batch normalization stabilizes training by normalizing the output of the previous layer, reducing internal covariate shift.

Upsampling Layers:

First Upsampling (Red): The tensor is then upsampled using transposed convolutional layers. The first upsampling layer increases the spatial dimension while reducing the depth (from 512 to 256). This process is visualized as the tensor expanding in width and height.
Additional Upsampling: Subsequent upsampling layers further increase the spatial resolution and continue to reduce the depth. Each layer generally doubles the width and height of the tensor while halving its depth.
Final Image Formation (Dark Blue): The last layer of the generator network is another transposed convolution that reshapes the tensor to the desired output dimensions. For RGB image generation, this means setting the depth to 3, corresponding to the three color channels. The spatial resolution at this stage should match the intended output resolution (e.g., 64×64 pixels).
Activation: the final layer's activation function is a Tanh, which normalizes pixel values to the range [−1,1][−1,1].
Note: with each subsequent layer halving the number of feature channels starting from 512 down to 3, indicating RGB output. Batch normalization is applied after each transposed convolution except for the last layer.

Output: Produces 64×64 RGB image conditioned on the input attributes.

Discriminator:



Input: Receives an RGB image of size 64×64 and a binary attribute vector with num_classes=40.

Architecture

 Similar to the generator, it consists of a series of Conv2d layers that double the feature channels from 64 to 512. Batch normalization and LeakyReLU are applied after each convolution. The final output is passed through a sigmoid activation function to classify the images as real or fake.

Discriminator takes an RGB image with 3 channels (red, green, blue) of size 64x64 pixels as input. Alongside the image, it receives additional conditional attributes (40). These attributes are reshaped and expanded to match the image dimensions and then concatenated along the channel dimension, augmenting the input tensor from 3 channels to 3 + num_classes=40. This allows the Discriminator to condition its classification on both the image data and the supplemental attribute information.
As the combined image-attribute data passes through the Discriminator's layers, it undergoes a series of transformations. The first convolutional layer expands the channel depth to 64 while halving the image dimensions to 32x32 pixels. Batch normalization follows each convolutional layer, standard practice for normalizing the data flow and accelerating the training process. LeakyReLU activation functions introduce non-linearity after each normalization step, with a slight slope for negative values to preserve gradient flow for negative inputs. Subsequent convolutional layers continue to double the channel depth while reducing the image dimensions, finally arriving at a depth of 512 channels with a spatial resolution of 4x4 pixels. A final convolution collapses this down to a single value per image, which is then flattened and passed through a sigmoid activation function to produce a scalar between 0 and 1, representing the Discriminator's assessment of the image's authenticity.

Output: A single scalar value indicating the authenticity of the input image.
Training Procedure

Epochs: 85
Batch Size: 64
Optimizers: Adam for both the generator and discriminator with a learning rate of 0.0004.
Loss Function: Binary Cross-Entropy (BCE) Loss to measure the performance of both discriminator and generator.

Generator Training

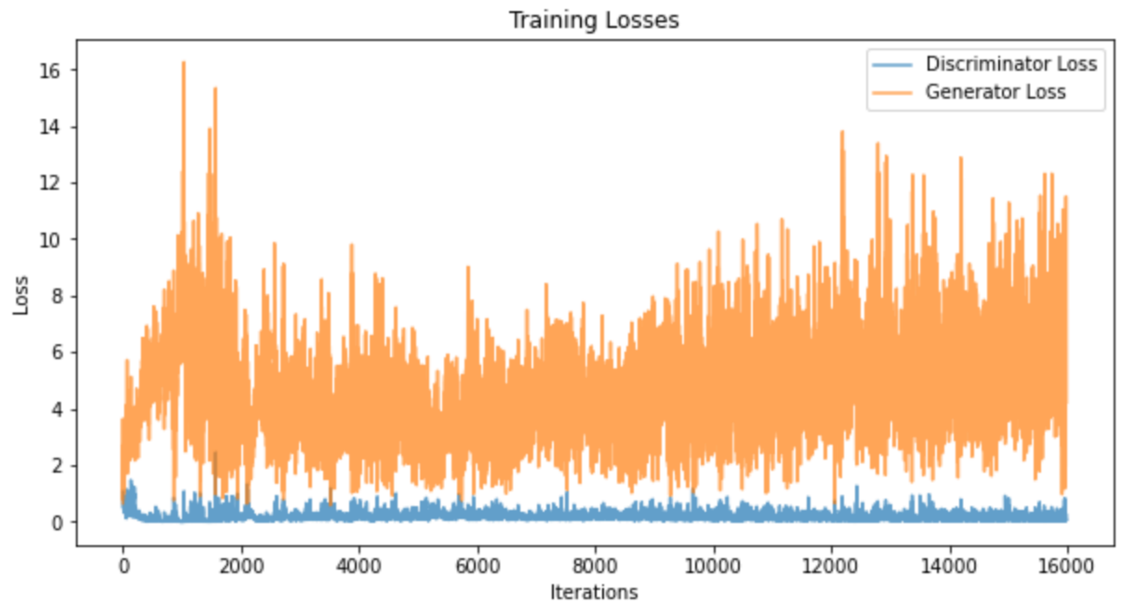
The generator is trained by attempting to fool the discriminator with fake images it generates.
The generator loss is computed by how well it tricks the discriminator into classifying the fake images as real.

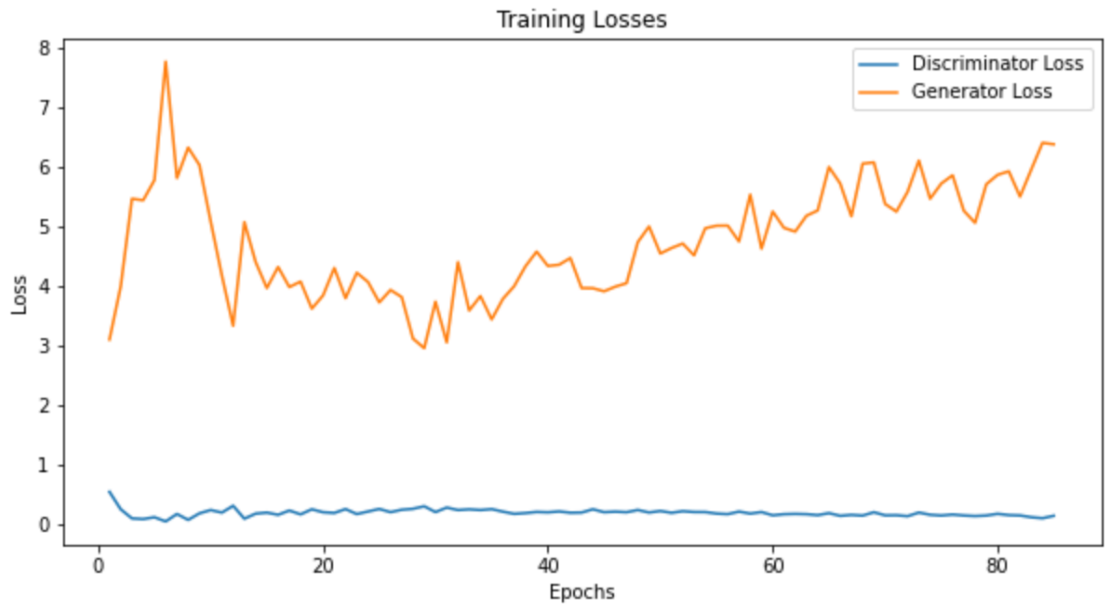
Discriminator Training

Trains on both real images with labels set to 1 and fake images with labels set to 0.
The discriminator loss is an average of the losses from real images (how well it recognizes real images) and fake images (how well it recognizes fake images).

Training Convergence Plots

The plots show the discriminator and generator losses over time, the discriminator seems to be consistently improving its ability to distinguish real images from fakes as indicated by the decreasing loss. The generator's loss is higher and shows more variance, indicating its struggle to produce convincing images that can fool the discriminator. As the training progresses, the generator should ideally improve, leading to a lower loss and less variability in the loss.

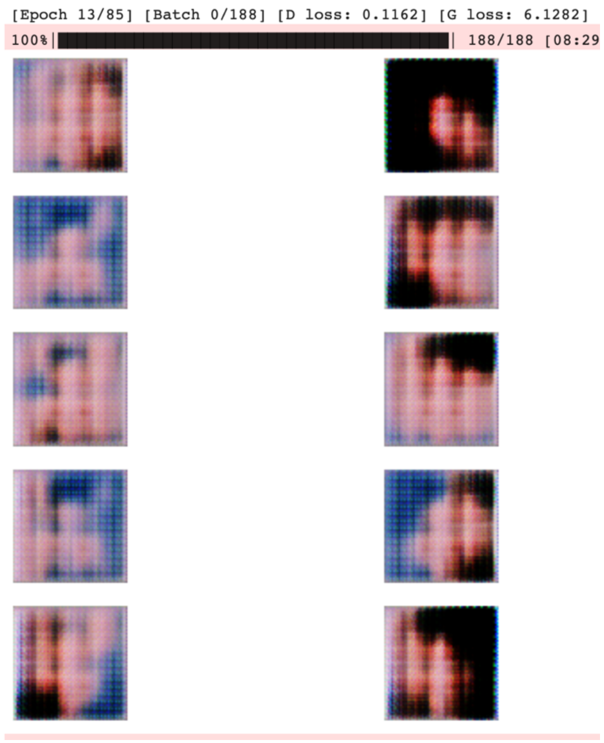




Summary of Attempts and Conclusions:

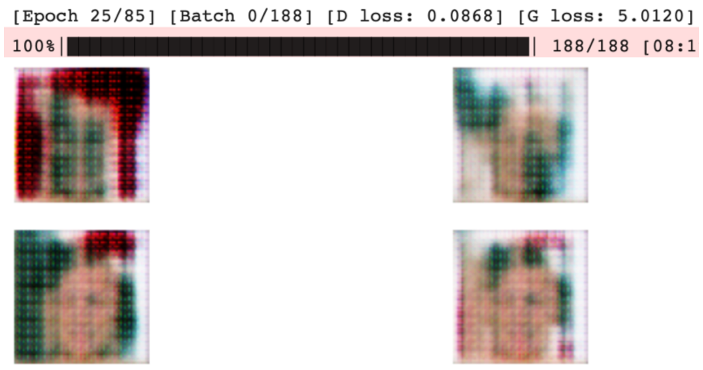
Your GAN has undergone various iterations to reach the current architecture. The initial large fluctuations in loss suggest that the model might have been unstable at the start, which is common with GANs due to the adversarial training dynamics. Over time, the generator's ability to create convincing images appears to have improved, as the discriminator's loss has decreased and stabilized, which is a good indicator of training progress.

This is the generated images with random attributes at epoch 13



```
[Epoch 13/85] [Batch 0/188] [D loss: 0.1162] [G loss: 6.1282]
100%|██████████████████████████████████████████| 188/188 [08:29
```

This is the generated images with random attributes at epoch 25



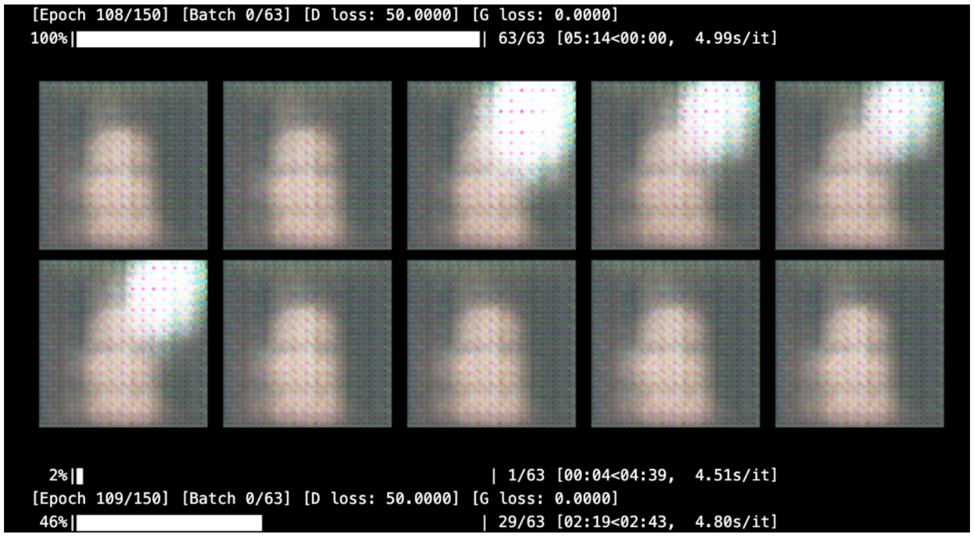
```
[Epoch 25/85] [Batch 0/188] [D loss: 0.0868] [G loss: 5.0120]
100%|██████████████████████████████████████████| 188/188 [08:1
```

This is the generated images with random attributes at epoch 41

```
[Epoch 41/85] [Batch 0/188] [D loss: 0.5632] [G loss: 2.1532]
100%|██████████████████████████████████████████| 188/188 [07:4
```

The generator can produce diverse faces, although they appear to be somewhat blurry and lack sharp features. This is common in GANs, especially in earlier epochs of training or when the model architecture and hyperparameters are not fully optimized.

In addition I tried some attemps before the last version of GAN, I had a miscalculation in dimention and didn't include the attributes of the images, in 108 epochs I have these images:

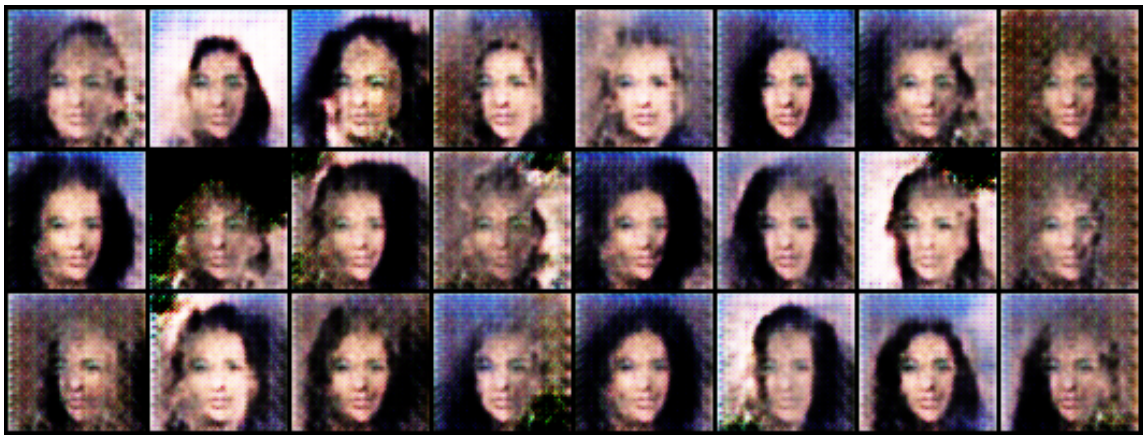


```
[Epoch 108/150] [Batch 0/63] [D loss: 50.0000] [G loss: 0.0000]
100%|██████████████████████████████| 63/63 [05:14<00:00,  4.99s/it]

  2%|▌                             | 1/63 [00:04<04:39,  4.51s/it]
[Epoch 109/150] [Batch 0/63] [D loss: 50.0000] [G loss: 0.0000]
 46%|██████████████               | 29/63 [02:19<02:43,  4.80s/it]
```

At last , with my final version of conditional GAN, it took approximtly 9 minutes for epoch, so I did only 85, although i can conclude from the proccess, with more epoches I get more image accuracy , but at the end and with the time limits I got these results :

```
This is generating images with these attributes : 'Attractive',
'Heavy_Makeup', 'High_Cheekbones', 'No_Beard', 'Wavy_Hair',
'Wearing_Lipstick','Young'
```



Conclusions Based on Results:

The discriminator’s performance improved over time, suggesting that it is learning to better distinguish between real and fake images.
The generator exhibits high variance in its loss, which could be due to various factors such as an imbalance in the training of the discriminator and generator, the need for more training epochs, or a need for hyperparameter tuning.
The quality of generated images suggests that while the model has learned to generate face-like structures, it may benefit from further refinement of training process (more epochs).