DATS Machine Learning II
Dr Amir Jafari

Face Image Generation

Mohammad Maaz, Suhas Buravalla, Carter Rogers

Introduction

Our final project involves using generative deep learning to generate human facial images. We selected this problem because synthetic generation of image data is a highly sensitive topic in the fields of surveillance and intelligence; according to the Associated Press, U.S. lawmakers first met to discuss the issue of artificially generated imagery in 2019[reference?]. The prevalence of synthetic image data has only increased since. We implement three model architectures as follows.

1. Variational Autoencoder
2. Generative Adversarial Network
3. Wasserstein Generative Adversarial Network

We will compare the results of the three models and select the best generated human facial images.

Dataset

For this task, we use the CelebA dataset which contains over 200,000 celebrity facial images each with 40 attribute annotations. These images cover huge variations in poses and the background clutter. The dataset contains over 10,177 identities, 202,599 images of faces and 5 landmark locations with 40 binary attributes annotations per image.

Generative Adversarial Networks

Generative adversarial networks (GANs) are a type of deep learning algorithm that is designed to generate new data that is similar to a training dataset.

Architecture

The architecture of GANs typically consists of two neural networks: a generator and a discriminator. The generator network is trained to generate new data that is similar to the training data, while the discriminator network is trained to distinguish real data from fake data. The two networks are trained together in a zero-sum game, where the generator tries to produce data that the discriminator cannot distinguish from the real data, and the discriminator tries to correctly identify the fake data produced by the generator. The pseudocode for this algorithm is as follows.

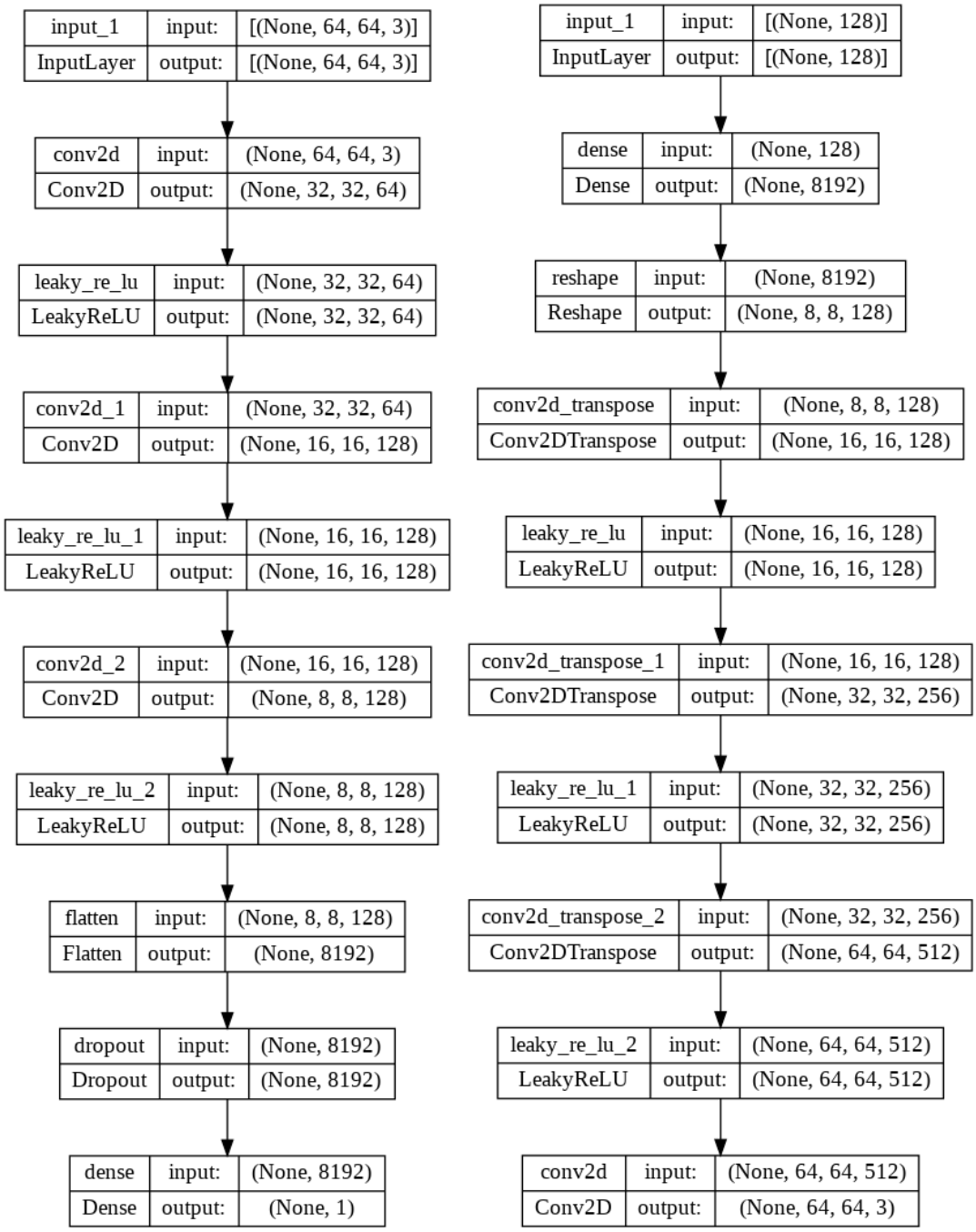
1. Initialize the generator and discriminator networks
2. For each batch of real images:
    a. Generate a batch of fake images using the generator
    b. Train the discriminator on the fake and real data

c. Train the generator on a batch of noise data, using the output of the discriminator as feedback

3. Repeat this process until the generator produces data that is indistinguishable from the real data

Network

The discriminator (left) and generator (right) networks are defined as follows.



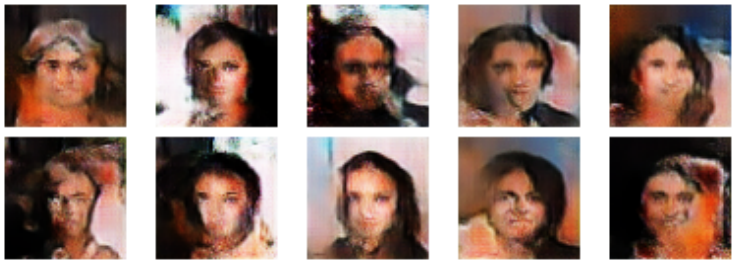
Experiment

I train the model on the CelebA dataset. For preprocessing, I resize the images to 64x64 pixels and divide them by 255 to get floating points for pixel values. I then run the model on 30 epochs. One epoch takes approximately 40 minutes to complete, so the experiment took around 20 hours. I implemented a custom callback that tests the generator after every epoch, allowing us to visualize the generated images and keep track of discriminator and generator losses.

Epoch: 10 Disc loss: 0.65 Gen loss: 1.06

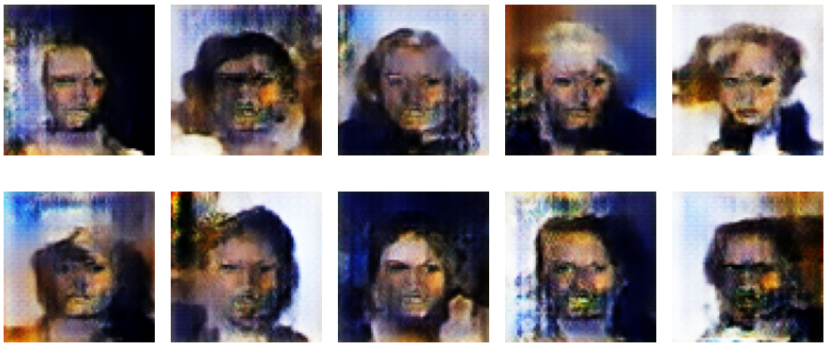
Epoch: 20 Disc loss: 0.60 Gen loss: 0.99

Epoch: 30 Disc loss: 0.55 Gen loss: 0.85



Results

The custom callback also saves the generator network periodically, allowing us to run inferences when we want. The results for the generator after 50 epochs are as follows.



As we can see, the images are distorted versions of human faces. It is quite easy to tell real and generated images apart.

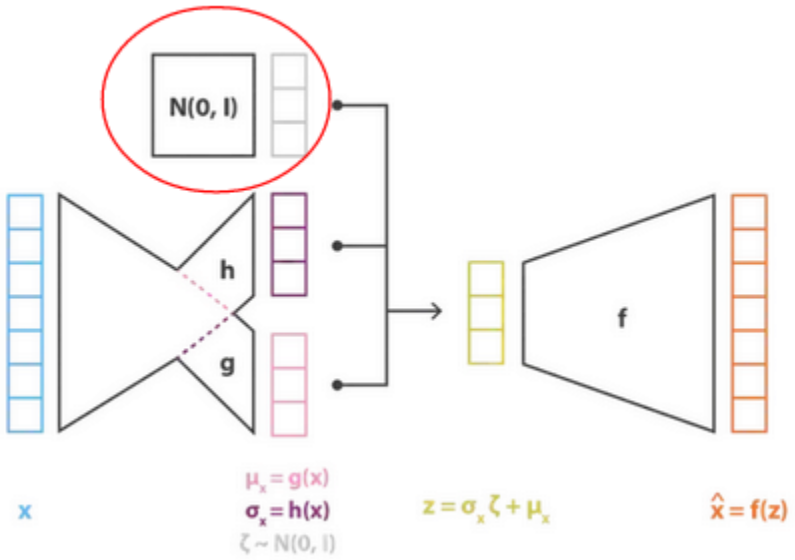
Variational Autoencoders

Architecture

Variational Autoencoders (VAEs) are similar to GANs in their ability to generate artificial image, text, and audio data, but the process by which the model generates the data is quite different. VAEs utilize an encoder/decoder structure similar to GANs, but a key aspect to VAEs is their dimensionality reduction by the encoder to preserve the maximum amount of information possible. Likewise, the object of the decoder is to minimize error in the reconstruction of the data (images, in the case of this project). The autoencoder (encoder/decoder) process is such that the data is fed through the encoder and decoder, comparing the output of the decoder to the input, and backpropagating the error over the network to update the weights accordingly. The autoencoder architecture ensures that only critical information is passed through and reconstructed through both Principal Component Analysis (PCA) and gradient descent to minimize error.

VAEs differ from traditional autoencoders in that the latent (encoded) space must be regularized in order to generate meaningful outputs. The autoencoder is trained to minimize loss,

but does not organize the latent space during training. Unless the network is regularized, the autoencoder will overfit as much as possible. We define VAEs as “an autoencoder whose training is regularized to avoid overfitting” (Rocca). The autoencoder of a VAE is trained to minimize reconstruction error. This is accomplished by the encoder returning a distribution over the latent space instead of a single point.



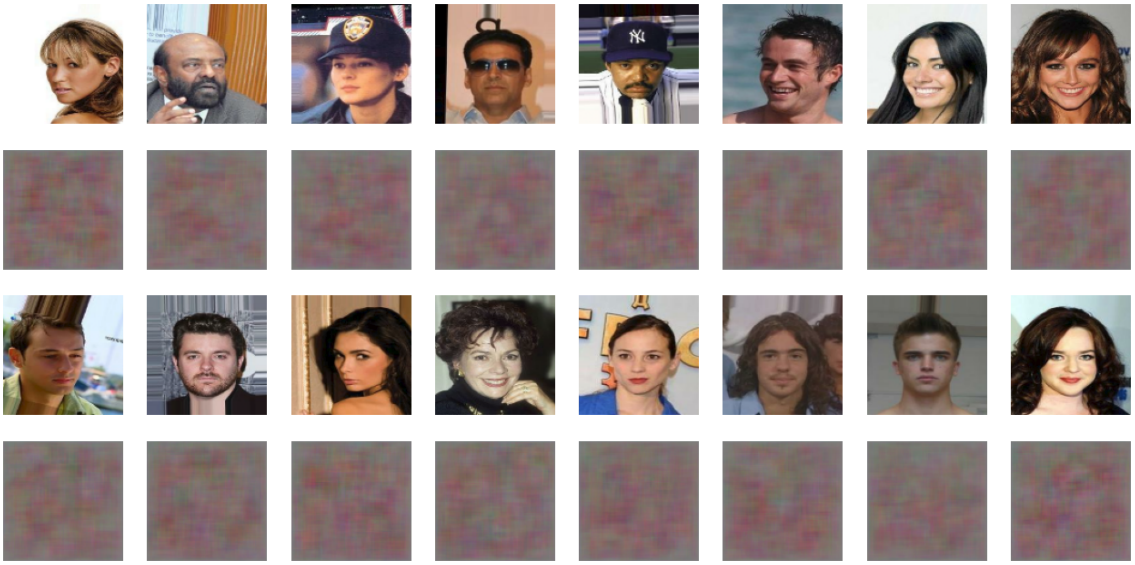
Architecture of a VAE network with Regularization term circled in red (Rocca).

Experiment

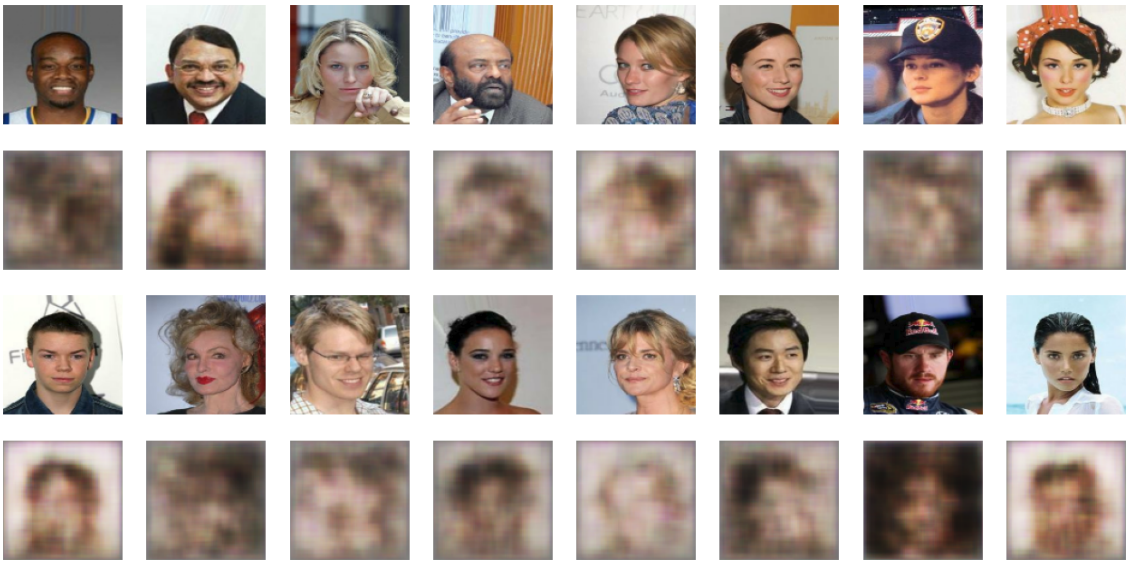
The experiment of generating facial images using VAEs follows a very similar process to that of using GANs. The images are resized to 112x112 with the pixels divided by 255. The training of the VAE took far less time than the GAN; using 50 epochs, the training was complete in roughly 100 minutes, for approximately 2 minutes per epoch. Compare this with the training time for the GAN, which took approximately 40 minutes per epoch. Thus, there was no need to downsize the images to a lower resolution, since training took a reasonable amount of time.

Results

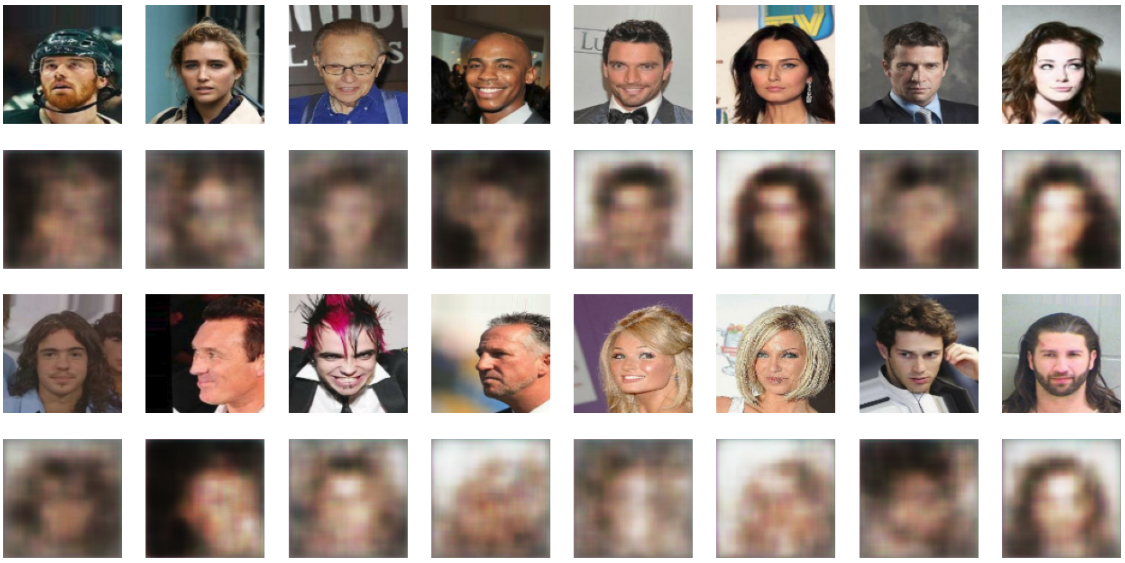
1 epoch:



20 epochs:



50 epochs:



One major benefit to using VAEs over GANs for image generation is the computational power needed for training. It is much easier to train a VAE network as opposed to a GAN. However, we observe that the images generated by the VAE network lack definition after 50 epochs of training. The outlines of the faces are visible, and hairstyles can be discerned, but the faces themselves appear uniform. This is one major weakness of VAEs as opposed to GANs; the images generated are less defined than those generated by the GAN. The regularization process of the VAE often leads to more “smoothing” of the images, resulting in faces with smooth outlines and less detail in the eyes, ears, nose, etc.

Wasserstein Generative Adversarial Networks

Overview, Architecture and Implementation

The idea of working of GANs is that there are two probability distributions, Probability distribution of the output of Generator and the Probability distribution of the real images and we want to minimize the gap between these two distributions. There are 3 ways to calculate the distance between these two distributions : Kullback–Leibler divergence, Jensen–Shannon divergence, and Wasserstein distance. The Jensen-Shannon divergence is typically used in GANs.

WGANs (K) use the Wasserstein distance to calculate the distance between the probability distributions and its architecture is similar to the architecture of the GANs. It consists of two neural networks: a generator and a discriminator. The generator network is trained to generate new data that is similar to the training data, while the discriminator network is trained to distinguish real data from fake data. The two networks are trained together in a zero-sum game,

where the generator tries to produce data that the discriminator cannot distinguish from the real data, and the discriminator tries to correctly identify the fake data produced by the generator.

In WGANs the discriminator acts as a critic. The reason behind this convention is that there is no sigmoid function to identify whether an image is fake or real (0 or 1), instead the discriminator returns a value in that range of 0 to 1 which allows the discriminator to act less strictly as a critic.

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z)]$$

(Arjovsky et al.) In the equation above, the max value is a constraint on the discriminator. The discriminator wants to maximize the distance between the real and generated data in order to distinguish between real and fake successfully, the generator tries to minimize the distance between real and generated data.

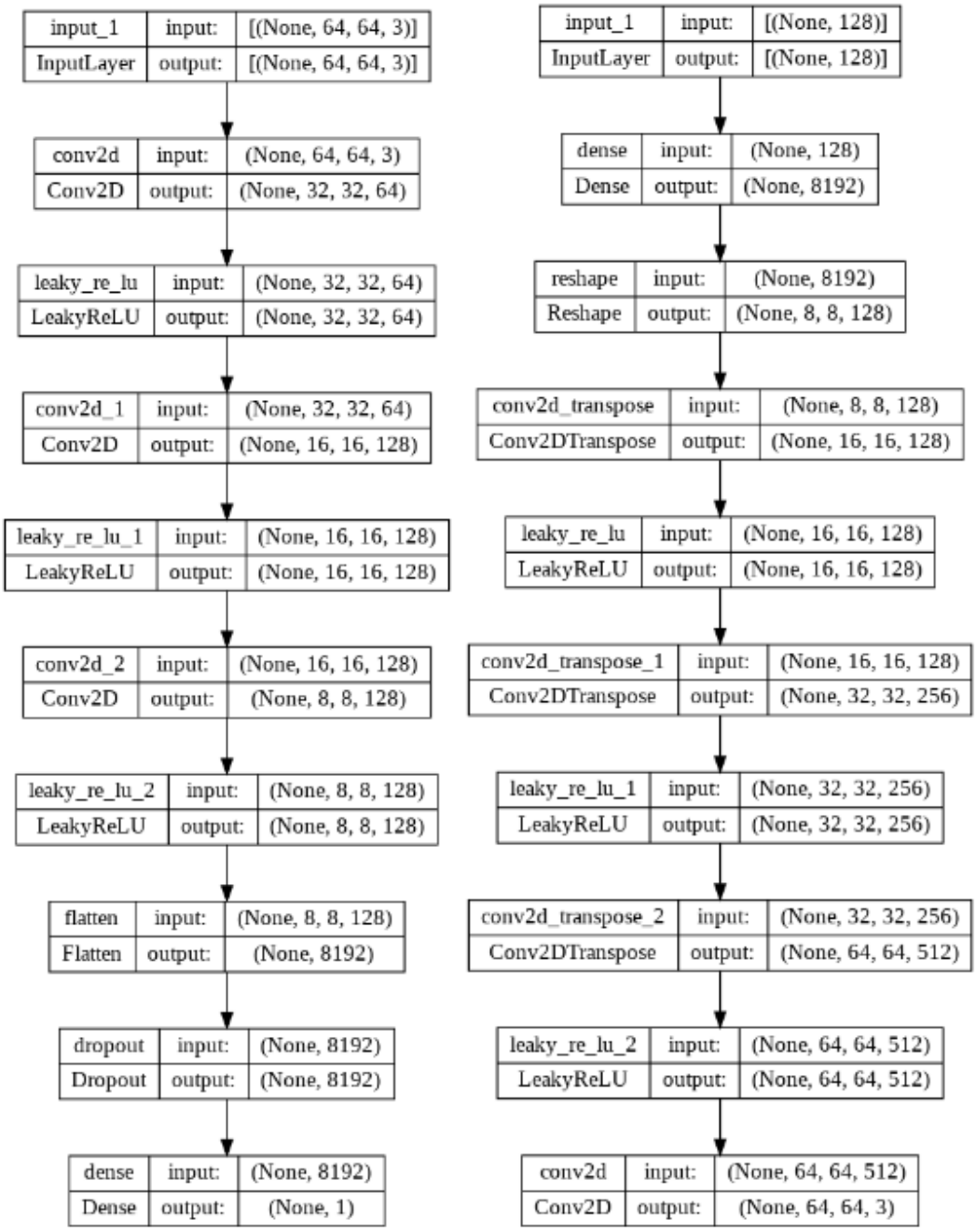
Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

(Gulrajani et al.)

The method proposed above is an alternative to weight clipping. The author proposes a Gradient Penalty by adding a loss term that keeps the discriminator gradient close to 1.

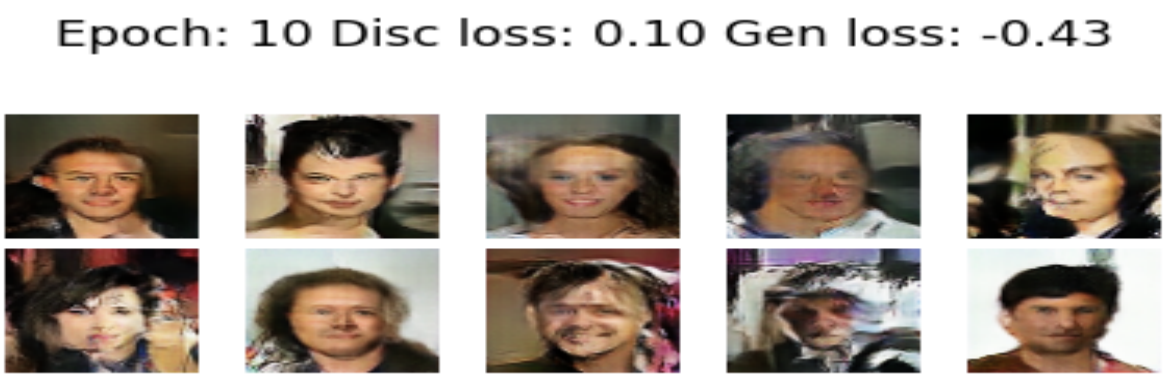
The lambda is the gradient penalty coefficient and n-critic is the number of critic iterations by the discriminator for each generator iteration. Alpha and beta values represent the constraints of the Adam optimizer.

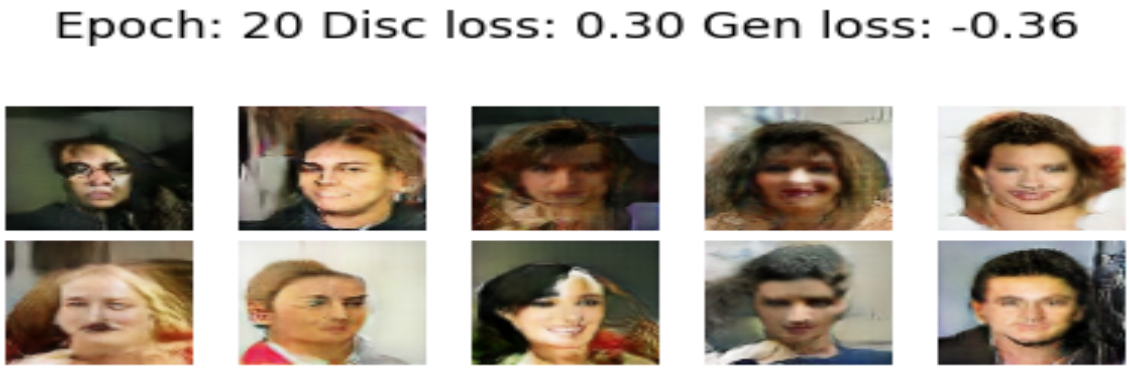


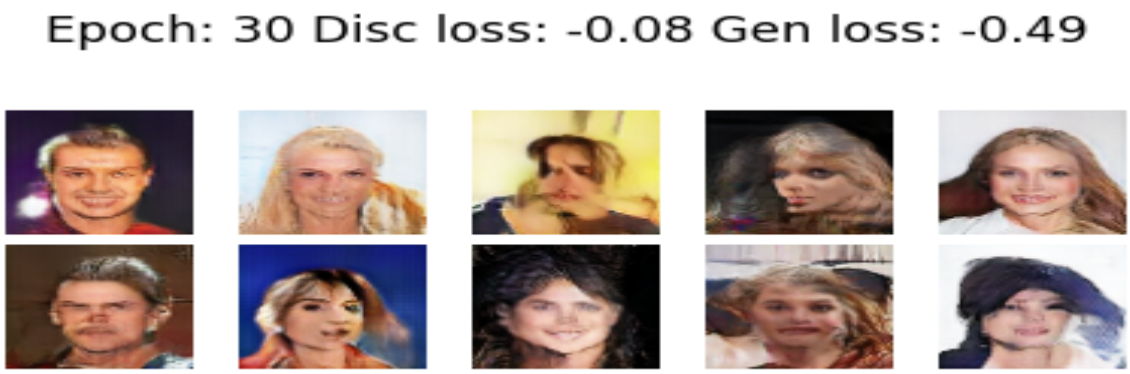
Experiment

Similar to the GANs, we resize the images to 64x64 pixels and divide them by 255 to get floating points for pixel values. We set the Gradient penalty to The model is run for 30 epochs and each epoch takes approximately 70 minutes, so the experiment took around 35 hours to complete. We implemented a custom callback that tests the generator after every epoch, allowing us to visualize the generated images and keep track of discriminator and generator losses.

Results







We can see that the edges of faces and some other features such as noses, eyes, teeth are highlighted better compared to the other networks.

Summary and Conclusion

Summarize the results you obtained, explain what you have learned, and suggest improvements that could be made in the future.

References

Cheong, Soon Yau. *Hands-On Image Generation with TensorFlow: A Practical Guide to Generating Images and Videos Using Deep Learning*. Packt Publishing, 2020.

Chollet, Francois. “DCGAN to generate face images.” *Keras*, 29 April 2019, https://keras.io/examples/generative/dcgan_overriding_train_step/. Accessed 12 December 2022.

Chollet, Francois. “WGAN-GP overriding `Model.train_step`.” *Keras*, 9 May 2020, https://keras.io/examples/generative/wgan_gp/. Accessed 12 December 2022.

Joseph, Rocca. “Understanding Variational Autoencoders (VAEs) | by Joseph Rocca.” *Towards Data Science*, 23 September 2019, https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73. Accessed 12 December 2022.

Liu, Ziwei. “CelebA Dataset.” *MMLab@CUHK*, https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html. Accessed 12 December 2022.