DATS Machine Learning II
Dr Amir Jafari

Face Image Generation - Individual Report

Suhas Buravalla

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

I implemented the Wasserstein Generative Adversarial Network.

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.

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 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.

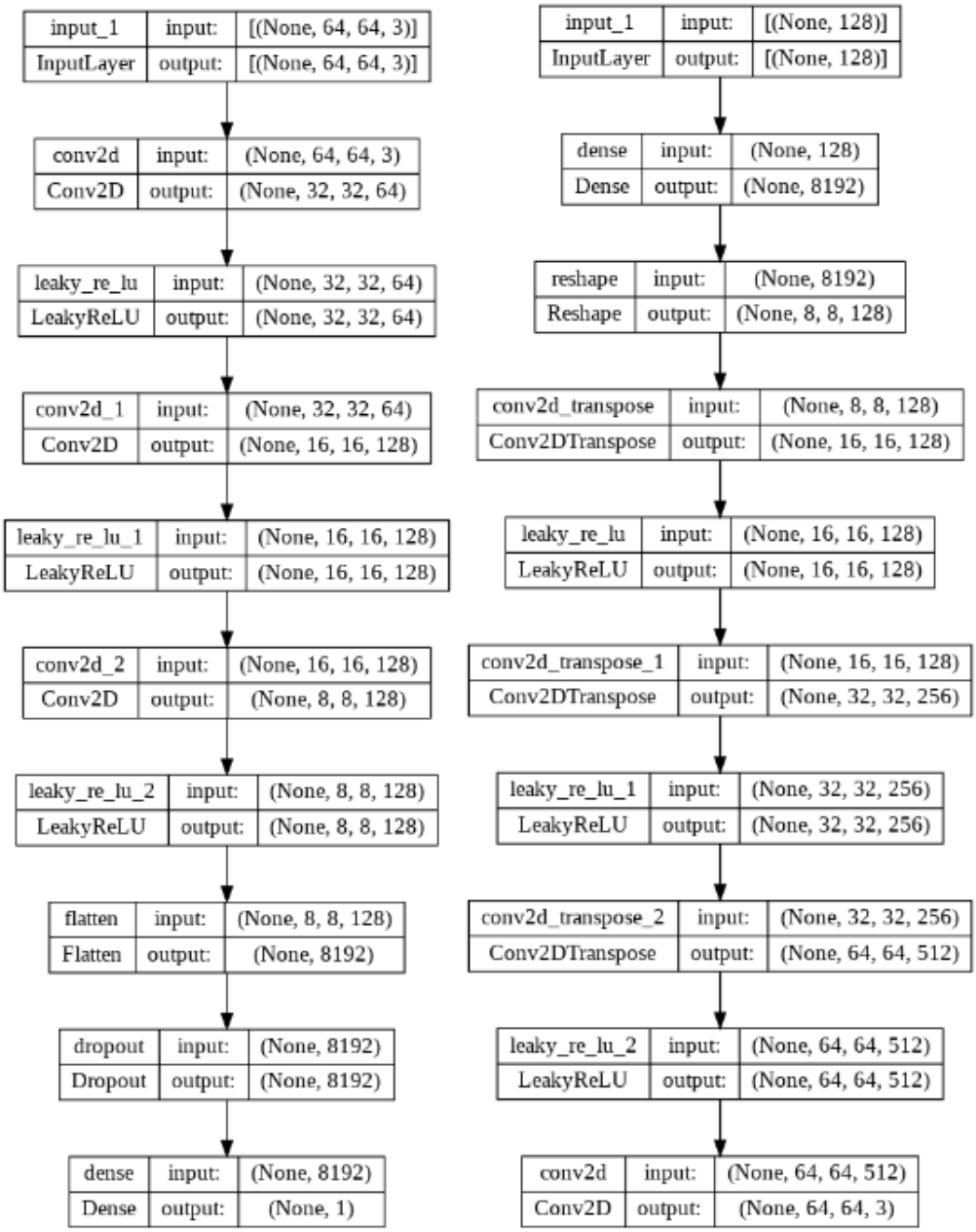
Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

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

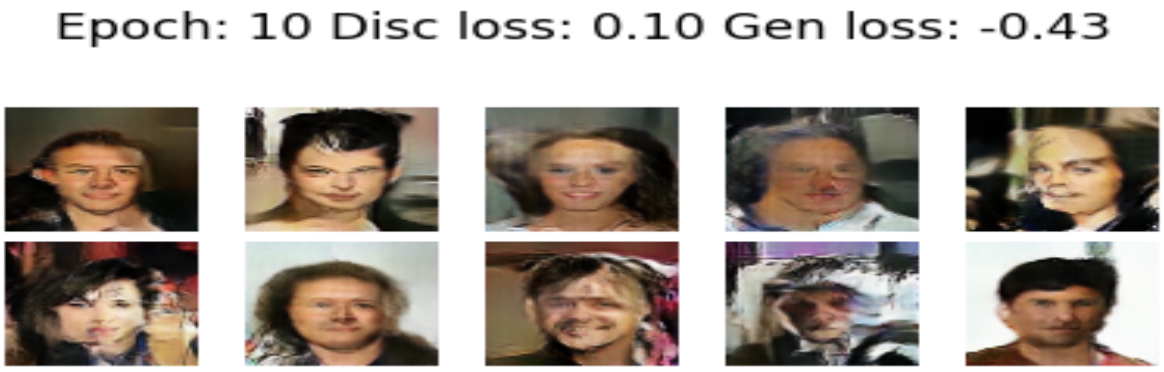
each generator iteration. Alpha and beta values represent the constraints of the Adam optimizer.

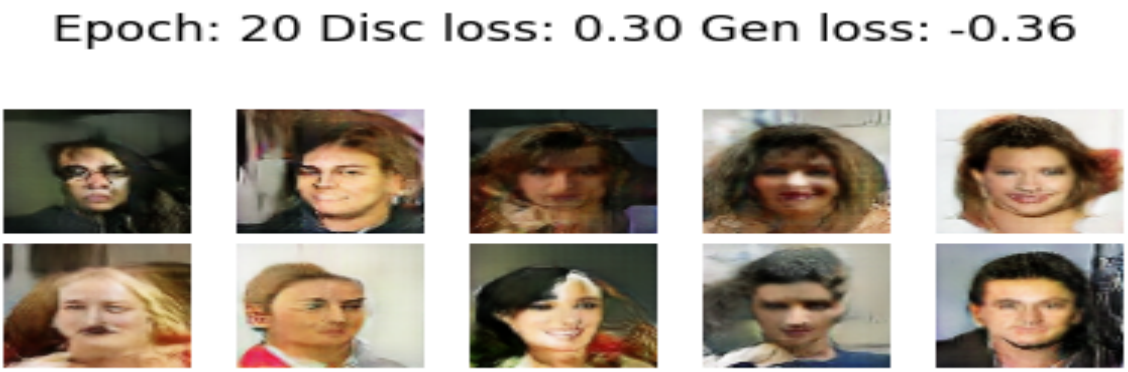


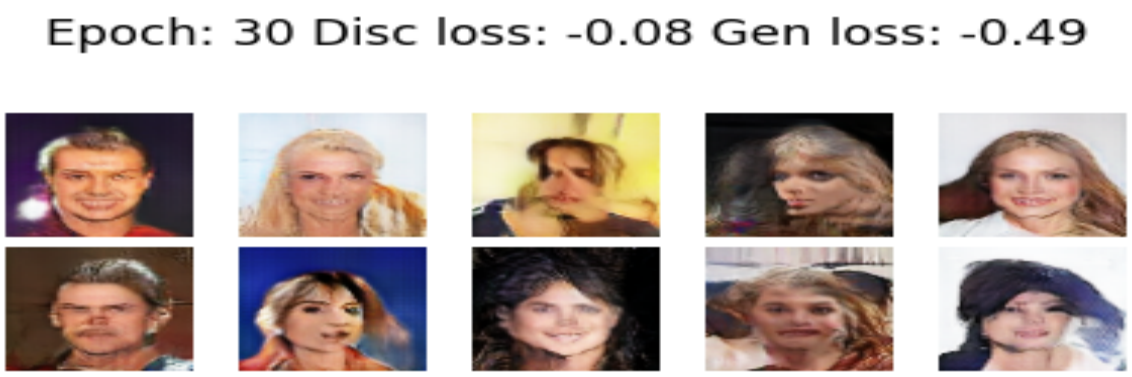
Experiment

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.

Problems faced during training

The way the Gradient Penalty is calculated generally in GANs or regular WGANs proved to be ineffective and resulted in NAN values for the generator and discriminator loss. The following fix was implemented after some research :
# 3. Calculate the norm of the gradients.
norm = tf.sqrt(tf.reduce_sum(tf.square(grads), axis=[1, 2, 3]))

Was changed to

# 3. Calculate the norm of the gradients.
norm = tf.sqrt(tf.reduce_sum(tf.square(grads), axis=[1, 2, 3])+1e-12)
This is because while back-propagating, tf.sqrt() is non-differentiable at 0, if the gradients of last batch are all 0, the norm would be infinite.

Summary and Conclusion

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. Wasserstein Generative Adversarial Network does provide better results when compared to GANs and VAEs.

Percentage of code

((111- 29)/ 111+ 24) * 100 = 60.74 %

References

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

Bharat K, “WGAN: Wasserstein Generative Adversarial Networks”, https://blog.paperspace.com/wgans/, Accessed December 10 2022

Martin Arjovsky, Soumith Chintala, L´eon Bottou, “Wasserstein GAN”, https://arxiv.org/pdf/1701.07875.pdf, Accessed December 10 2022

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville, “Improved Training of Wasserstein GANs”, https://arxiv.org/pdf/1704.00028.pdf, Accessed December 10 2022