Facial Expression Synthesis by the Conditional Generative Adversarial Network

Edoardo Mortara id. 954198
Affective Computing a.a. 2020/2021
University of Milan

1 Facial Expression Synthesis

Rendering photo realistic facial expressions from single static faces while preserving the identity information is an open research topic which has significant impact on the area of affective computing. Generating faces of a specific person with different facial expressions can be used for various applications including face recognition, face verification, emotion prediction, expression database generation, augmentation and entertainment.
In this paper the problem to be solved is to synthesize photo realistic faces with a specific facial expression. In particular the target facial expressions are: angry, disgust, fear, happy, neutral, sad, surprise.

2 State of the art

Facial expression synthesis is used for photo realistic face rendering and virtual avatar animation. Synthesis can be done by using Geometry-driven models [5] which extract geometric features or 3D meshes and maps them to avatars or new faces.

Another way for synthesis is Neural Networks; here some of the most recent and significant works.
Nejadgholi et al. [6] uses brain-inspired model to generate faces; Yeh et al. [7] uses a variational autoencoder to find a flow map to warp a source image in such a way that the target image has a specific facial expression; Zhou et al. [4] obtained the best results in the current state of the art by using CDAAE (Conditional Difference Adversarial Autoencoder) to model the correlations between different expressions and facial movements.

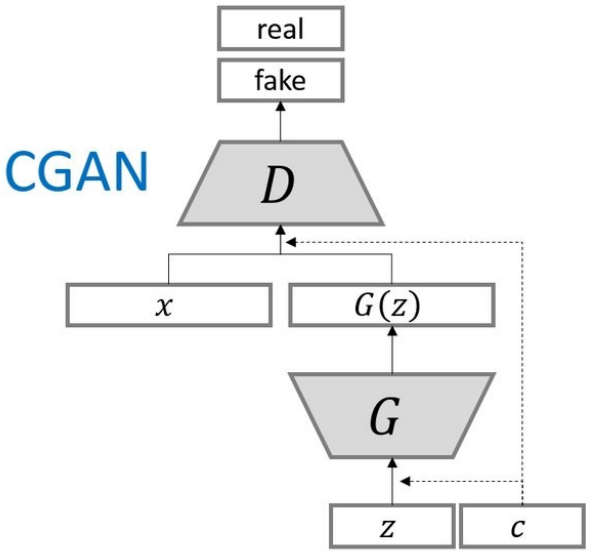

Figure 1: CGAN architecture

3 Generative Modeling

Generative modeling is one of the many approaches to unsupervised learning [1]. The training examples $x \in Data$ are picked from an unknown distribution $p_{Data}(x)$, and the goal of the generative model is to learn a new distribution $p_{Model}(x)$ that well approximates $p_{Data}(x)$.
The most used technique is maximum likelihood estimation: here the goal is to minimize the Kullback-Leibler divergence between p_{Data} and p_{Model}. This density modeling has been successfully used in traditional statistics, but

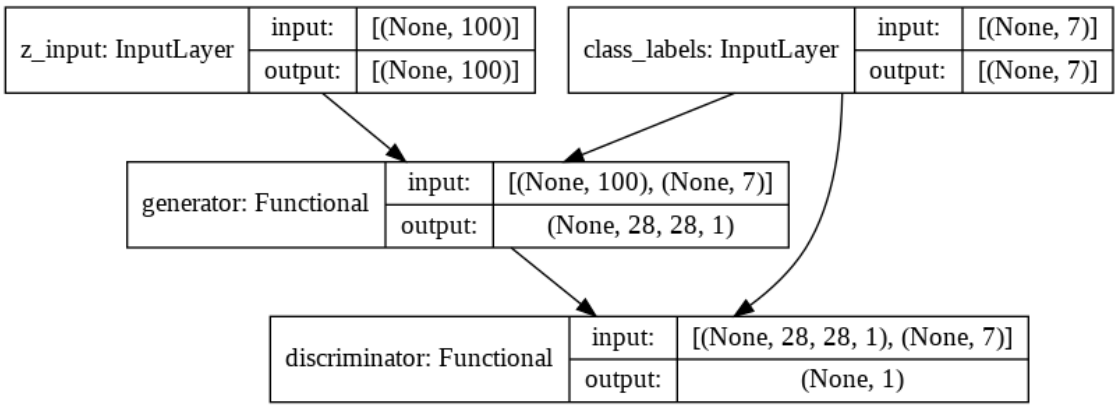

Figure 2: Adversarial network architecture

it's hard to use for complicated distributions.
For these reasons it's better to use an *implicit generative model*: we avoid to design a tractable density function and learn only a tractable sample generation process. GANs and variational autoencoders belong to this type of model.

4 Generative Adversarial Network

Generative adversarial networks (GANs) [1] achieved remarkable results in various computer vision tasks such as image generation, image translation, etc. The goal of GANs is to model distribution as similar as possible to the true data distribution. To achieve this goal, the generator G and the discriminator D of GANs compete in a two player minimax game. Specifically, the discriminator learns to distinguish real samples from fake samples while the generator learns to generate fake samples to fool the discriminator, until the Nash equilibrium [2] is reached between the two modules; this equilibrium is such that each player reach a local minimum of their cost functions with respect to that player's parameters. This mean no player can reduce its cost function assuming other player's parameters do not change.
The generator defines $p_{Model}(x)$ implicitly, without evaluating the density function p_{Model}. The generator is defined by a prior distribution $p(z)$ over a latent space vector z that serves as input to the generator function $G(z;\theta_G)$ (θ_G are the generator learnable parameters). The goal is to generate realistic samples from z.
The other player is the discriminator; it examines samples of $x \in Data$ and, with the discriminator function $D(x;\theta_D)$ (θ_D are the discriminator learnable parameters), verifies if they are real, or fake.
The players have a cost function: $J_G(\theta_G, \theta_D)$ for the generator and $J_D(\theta_G, \theta_D)$ for the discriminator. Usually the cost functions are such that $J_G = -J_D$; this way we define a minimax game between the generator and the discriminator.

5 Conditional GAN

The CGAN [3] (see Figure 1) is an extension of the GAN where the model receives additional variables (in this case label) as input, which control in a deterministic way the output of the generator. This mean that the labels allow us to explore latent space very easily. CGAN has been successfully applied to synthesize images from labels, reconstructing objects from edge maps, etc.
More precisely the generator function accept latent space z and a $label$; similarly the discriminator function accept $x \in Data$ and a $label$. As stated in [3] the objective function of the two-player minimax game will be:

$$min_{\theta_G} max_{\theta_D} [E_{x\sim P_{data}(x)}[logD(x|label)] + E_{z\sim P_z(z)}[log(1 - D(G(z/label))]]$$

In the Implementation section I will provide more details about players' cost function.

6 Data sets

The Databases used in this project are: FER-2013 (*www.kaggle.com/msambare/fer2013*) and MMAFEDB (*www.kaggle.com/mahmoudima/mma-facial-expression*); they are both open source and contain gray scale and colored pictures of seven different facial expression (angry, disgust, fear, happy, neutral, sad, surprise).
Since the databases are very similar I decided to merge them into a single database called "fuse". I made this decision, because the results obtained by training the neural network only with FER-2013 are really similar to the results obtained by training only with MMAFEDB.
The images are resized to 28x28x1 pixels (gray scale), transformed in Numpy array and finally saved in a Pickle file. A Label is an integer $\in [0, 6]$; the labels are saved as Numpy array in a Pickle file. Before training the labels must be one-hot encoded.
The neural network normalize the images in the [0,1] interval. I tried also [-1,1] normalization, but results were not improving.

7 Implementation

The implementation phase was characterized by many "trials and errors". The best results (see Results section) were obtained using the following configuration.

The Discriminator (Figure 7) is a convolutional neural network that accept a 28x28x1 image (gray scale) and a label (one-hot encoding of the seven labels). The label is transformed with a Dense layer into a tensor of shape 28x28x1; after this, I create a 28x28x2 tensor by concatenating the image and the label. This new input is processed by three convolution layers that are batch normalized (except the first one, so that the model can learn the correct mean and scale of the data distribution) and include maxout activations, also known as leaky RELU. The number of filters are: 32, 64, 128; all filters are 5x5 and the stride is always 2.
In the end the output is flattened and processed by a Dense layer with a sigmoid activation function. The final output will be a number between 0 (fake image) and 1 (real image).

The Generator (Figure 6) is a convolutional neural network that accept a 100-dim latent space z sampled in a uniform distribution between $[-1, 1]$, and a label (one-hot encoding of the seven labels). The input is processed by four transposed convolution layers (for upscaling) that are batch normalized and include leaky RELU. The number of filters are: 256. 128, 64, 1; all filters are 5x5 and the stride is 2 for the first two layers, then is set to 1.
The output of the generator is a 28x28x1 (gray scale) image.

The cost function used for the Discriminator is the cross-entropy:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = E_{x\sim P_{data}(x)}[logD(x|label)]$$

$$+E_{z\sim P_z(z)}[log(1 - D(G(z|label))]$$

The first term represents feeding the actual data to the discriminator, and the discriminator would want to maximize the log probability of predicting one, indicating that the data is real. The second term represents the samples generated by G. Here, the discriminator would want to maximize the log probability of predicting zero, indicating that the data is fake. The generator, on the other hand tries to minimize the log probability of the discriminator being correct. The solution to this problem is an equilibrium point of the game, which is a saddle point of the discriminator loss.
The cost function used for the Generator is the cross-entropy:

$$J^{(G)} = E_{z\sim P_z(z)}[logD(G(z|label))]$$

The generator will maximizes the log probability of the discriminator being mistaken. Now D and G are both conditioned by z, x and $label$.
The minimax game is defined by the minimax objective function:

$$min_{\theta_G} max_{\theta_D}[E_{x\sim P_{data}(x)}[logD(x|label)]$$

$$+E_{z\sim P_z(z)}[log(1 - D(G(z/label))]]$$

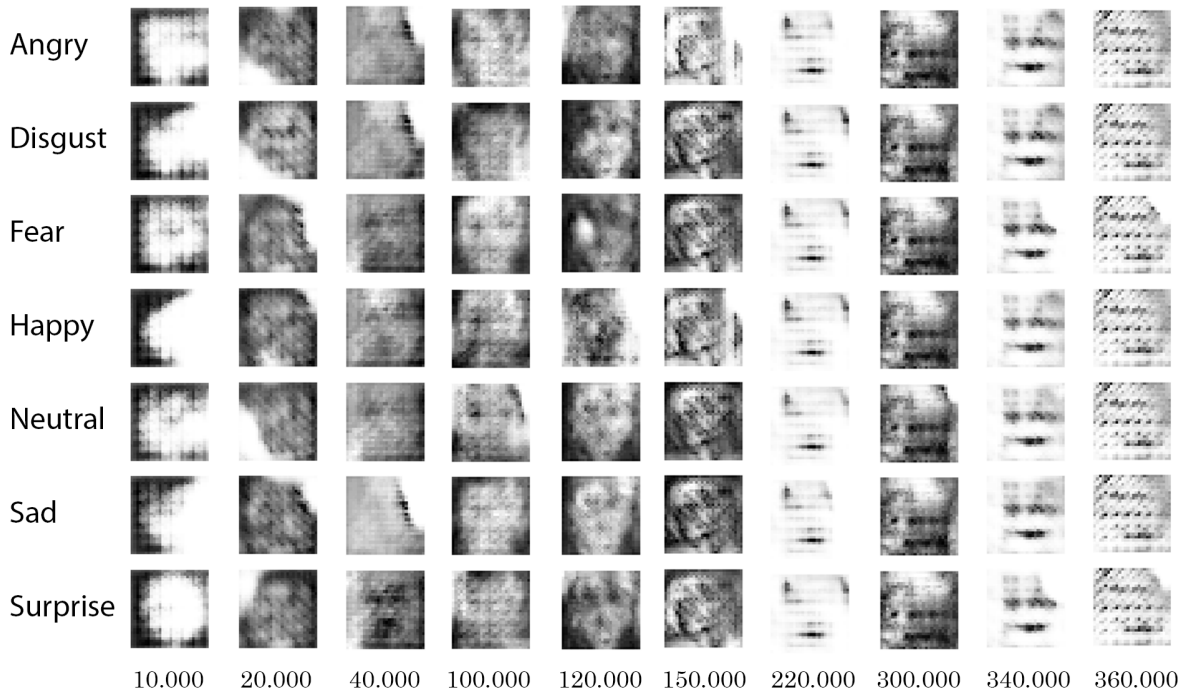

Figure 3: Generator output at different training step

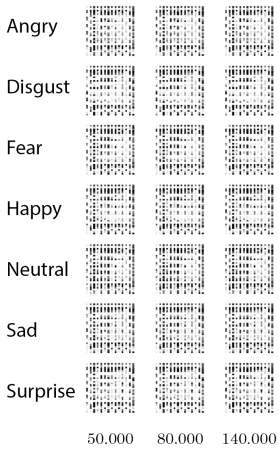

Figure 4: V1 generator results at different training step

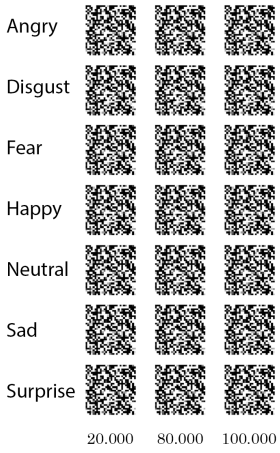

Figure 5: V2 generator results at different training step

The training process performs Stochastic gradient descent by sampling mini-batches of size 64. The first mini-batch of $x, label \in Data$ is used as input for the discriminator; here the discriminator goal is to learn that these are real images. Then we create a "fake" mini-batch using the generator and random labels for training the discriminator to learn that these are fake images.
The second mini-batch is used to train the Adversarial network (Firgure 2); here the mini-batch is composed by random z and random labels. The goal for the generator is to fool the discriminator; in the ideal case the discriminator can't distinguish real samples from fake ones and the loss is 0.5.

For both Generator and Discriminator I used the Adam optimization algorithm with the following configuration parameters:

- learning rate = 0.0002;
- beta 1 = 0.05;
- beta 2 = 0.09;

The other parameters are set to *Keras* implementation's default value (see *keras.io/api/optimizers*).

8 Results

The model proposed in the last section has been trained with SGD for 400.000 iterations. Figure 3 shows that the neural network has stopped learning after 150.000 iterations. The loss score of both discriminator and adversarial network converge to 0 after 50.000 iterations.
The reason for this bad learning is due to the fact that the discriminator becomes better than the generator; this mean that the generator fails to fool the discriminator, his loss drops to ~ 0 and its learning is slow.
Another interpretation of these results can be the "Mode collapse"; a mode collapse refers to a generator model that is only capable of generating one or a small subset of different outcomes, or modes.
I built two models to tackle these kinds of problems. In the first one, which we will call 'V1', the latent space is reduced from 100 to 80, and the discriminator is "impaired" using dropout layers and less filters.
As explained in [8] and [9] convergence failure happens when the generator and discriminator do not reach a balance during training. The second model, which we will call 'V2' is similar to V1 but stop the discriminator training after 50.000 iterations or when its accuracy is too high. As shown in Figures 4 and 5 not only V1 and V2 don't solve the Mode Collapse and the Convergence Failure, but they have even more problems.

References

[1] I. Goodfellow et al., "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst. (NIPS), Montreal, QC, Canada, 2014, pp. 2672–2680.

[2] J. F. Nash, Jr., "Equilibrium points in n-person games," Proc. Nat. Acad. Sci. USA, vol. 36, no. 1, pp. 48–49, 1950.

[3] M. Mirza and S. Osindero. (Nov. 2014). "Conditional generative adversarial nets." [Online]. Available: https://arxiv.org/abs/1411.1784

[4] Zhou, Yuqian & Shi, Bert. (2017). Photorealistic Facial Expression Synthesis by the Conditional Difference Adversarial Autoencoder.

[5] Zhang, Qingshan, et al. "Geometry-driven photorealistic facial expression synthesis." IEEE Transactions on Visualization and Computer Graphics 12.1 (2005).

[6] Nejadgholi, Isar, Seyyed Ali SeyyedSalehi, and Sylvain Chartier. "A brain-inspired method of facial expression generation using chaotic feature extracting bidirectional associative memory." Neural Processing Letters 46.3 (2017).

[7] R. Yeh et al, "Semantic facial expression editing using autoencoded flow," arXiv Preprint arXiv:1611.09961, 2016.

[8] *machinelearningmastery.com/practical-guide-to-gan-failure-modes*

[9] *www.mathworks.com/help/deeplearning/ug/monitor-gan-training-progress-and-identify-common-failure-modes.html*

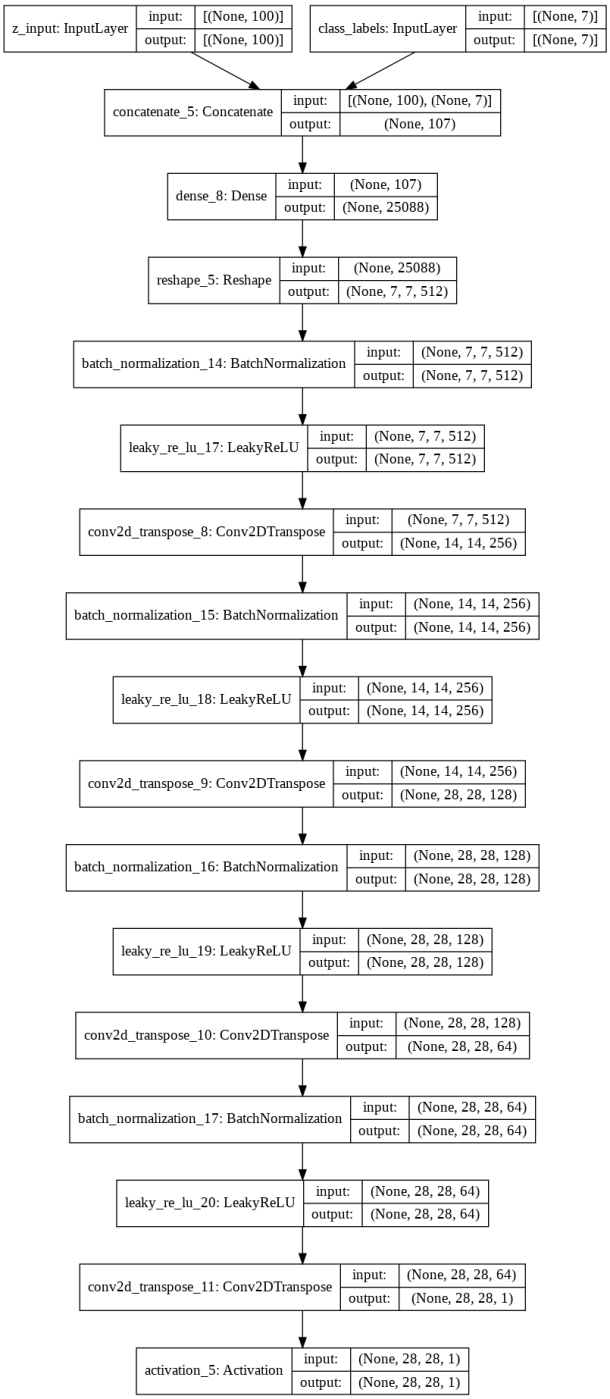

Figure 6: Generator architecture

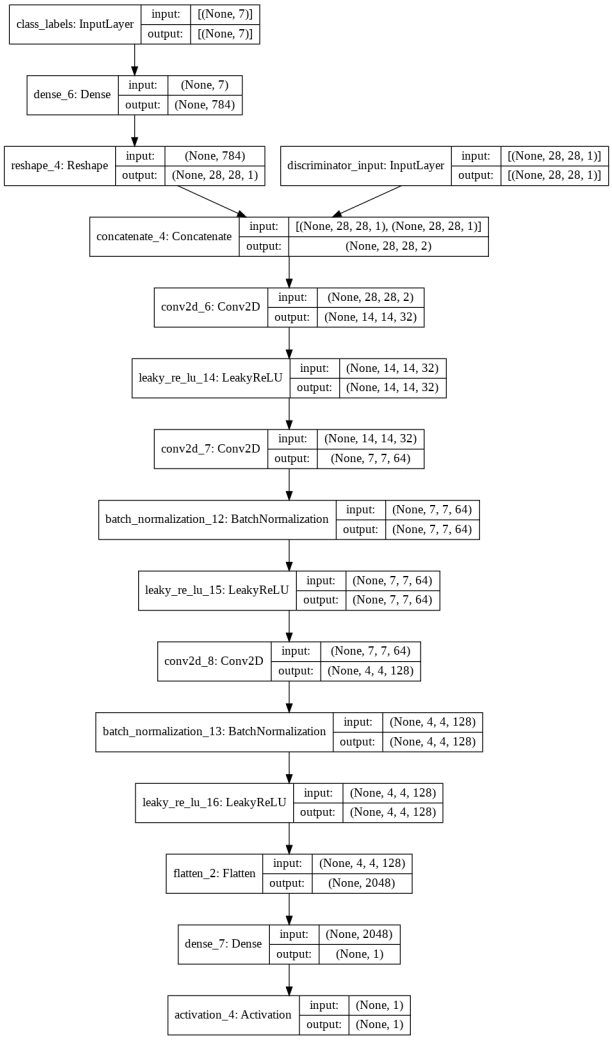

Figure 7: Discriminator architecture