## GAN 256x256

For the first milestone, I constructed the GAN model for the image, which is an image size of 256*256. In the training process, discriminator D gradually learns the latent perceptual structure inside the target images, and then the generator is trained on this (Goodfellow et al., 2014). The generator gives sub-optimal estimates that will be used as negative samples in the discriminator training. In the discriminator, a binary classification is performed with the grouped negative and positive (target images) samples, quantifying the latent perceptual distance between the two types of samples.

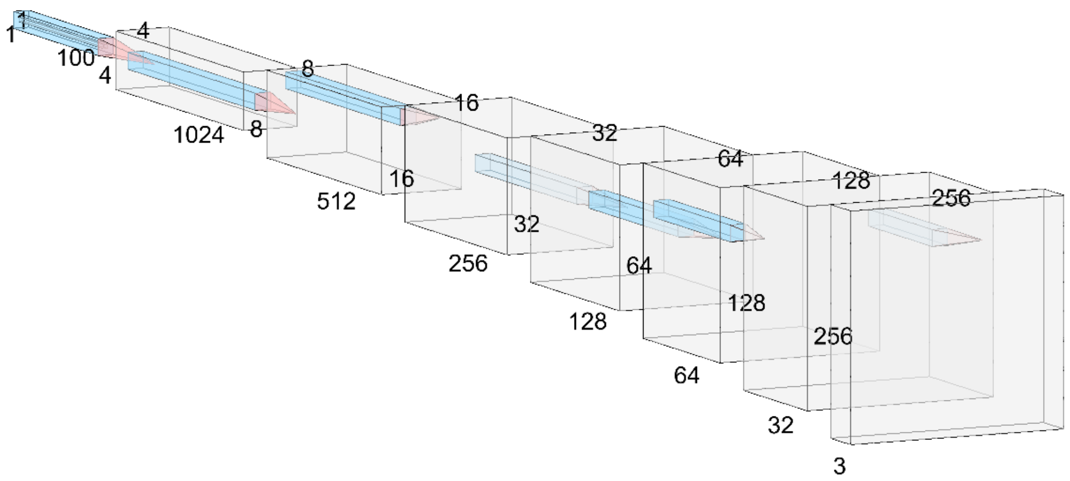
Gives the following notations: D, representing a discriminator; G, a generator. X and Y are random input signals and actual images from the training set. The generator and discriminator are in a two-player minimax game during the training process. This knowledge, the discriminator learns to distinguish between the estimated images G(X) and the actual images Y. The generator aims to reduce the gap between the estimated G(X) and the actual image Y. The training process is based on adversarial learning strategies, and the latent distribution inside the target images is learned and used. Lastly, the training will attain a state of equilibrium where the generated estimations of the generator will have the same latent perceptual space as the target images.

## Architecture of the Generator and Discriminator

GAN framework consists of generator G and discriminator D. In this section, the structures of the two networks will be described.

Structure of the Generator G

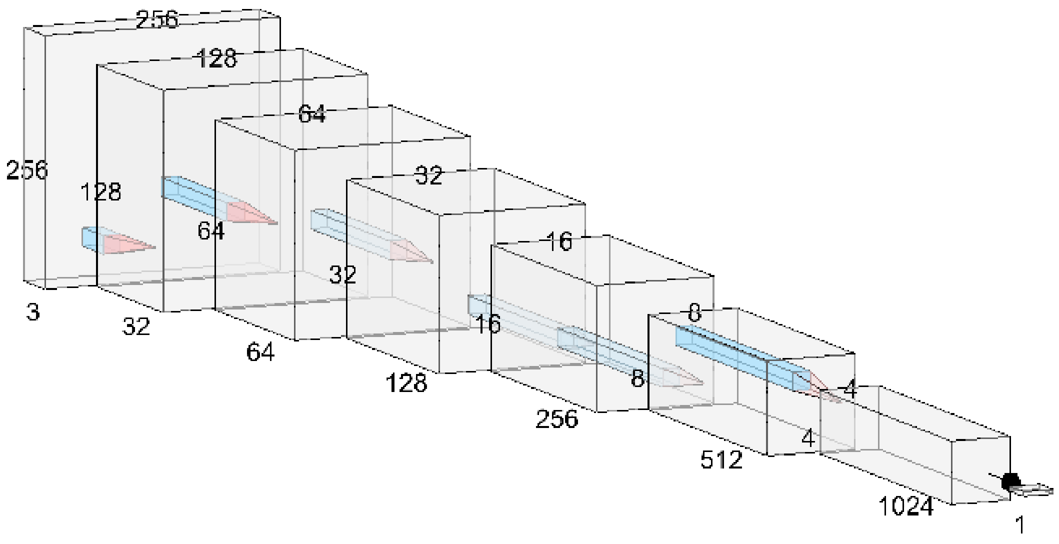
In this project, the generator G is to generate a face image G(X) with the size of 3×256×256 (the numbers are the orders of Channels, image Height, and image Width. The value 3 denotes the RGB channels) from a random latent vector X with the size of 100×1×1. [1]



To make the elaboration easier, the input of the generator is set to 100 random values to create the latent vector X of 100*1*1 size. The latent vector is then processed by a deconvolutional block (Deconv.) that contains a deconvolutional layer (filter size: It is followed by a convolution layer with 4×4 size, number of filters 1024, stride one and padding size 0, a batch normalization layer and ReLU activation function. After the process of the Deconv. Block, it gets the feature map of 512×8×8. Next, the feature map is processed by another Deconv. Block (filter size: 4×4, number of filters: 256, stride: 2, and padding size: 1), which upsamples the feature map to the size of 256×16×16. Likewise, the feature map is upsampled three more times. Finally, the generator generates the generated image G(X) with the size of 3×256×256.

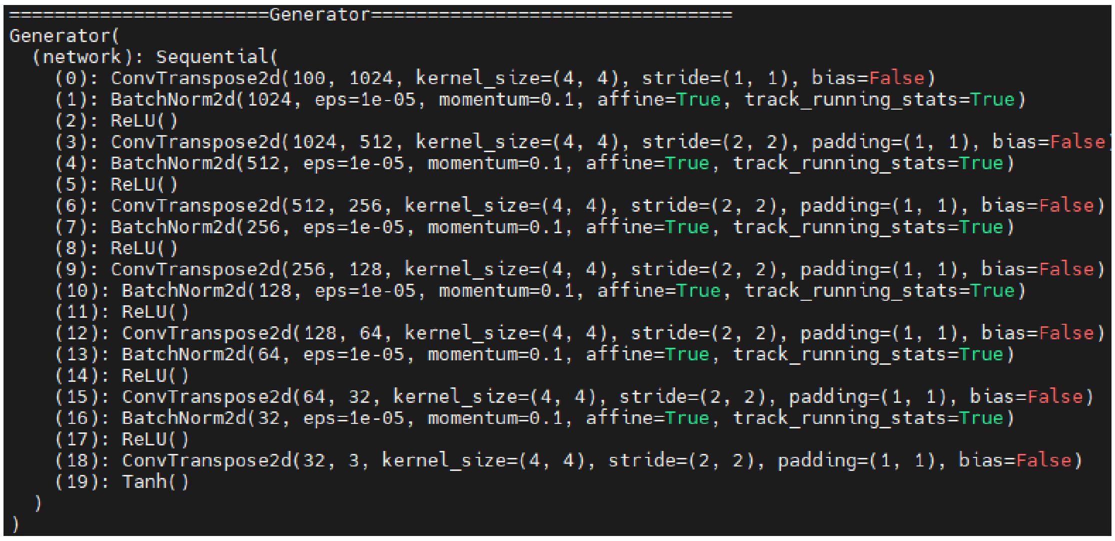
Structure of the Discriminator D

The discriminator D decides whether the face image Y is real or the generated (fake) face image G(X). In this lab, all face images are the same size of 3x256×256, where the value 3 represents the RGB channels. The output of the discriminator is the probability (size: It also includes a binary classifier (1×1×1) that the input image is an actual face image.[2]

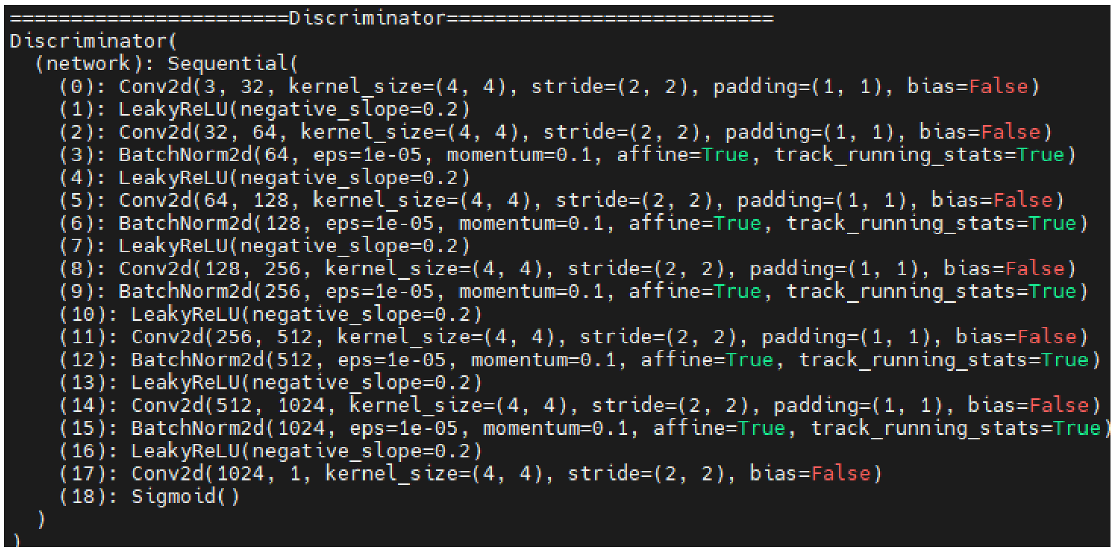


The input image can be the actual face image of training dataset Y or the fake face image generated by the generator G. The discriminator, on the other hand, seeks to determine what class the input image belongs to and the likelihood it is an actual face image. First, a convolutional block with a LeakyReLU activation function and a convolutional layer (filter size: For the initial first block, namely 4×4, number of filters 128, and stride two and padding size 1 operate on the input image. But that would reduce the spatial dimension to 128 x 32 x 32. The feature map subsequently goes through another convolution block. The final convolutional block with the sigmoid function predicts a single value (size: which indicates (1×1×1) to explain the probability of the input being an actual facial image. Two other convolutional blocks follow to perform the downsampling of features more effectively.

The result is as follows. Discriminator and generator layer information are also given to provide a better understanding of the architecture.[3]



```
======================Generator===============================
Generator(
  (network): Sequential(
    (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
    (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU()
    (15): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (16): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU()
    (18): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (19): Tanh()
  )
)
```



```
======================Discriminator==========================
Discriminator(
  (network): Sequential(
    (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2)
    (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2)
    (8): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2)
    (11): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (12): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): LeakyReLU(negative_slope=0.2)
    (14): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (15): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): LeakyReLU(negative_slope=0.2)
    (17): Conv2d(1024, 1, kernel_size=(4, 4), stride=(2, 2), bias=False)
    (18): Sigmoid()
  )
)
```

## Training Procedure

The Discriminator's Training Method The discriminator D distinguishes between the synthesized fake face images, G(X), and the real face images Y. This discriminator D provides a binary classification measure and outputs the probability that the input is an actual face image. The real face photos of the training data are given y = 1, and the fake images, synthesized using the generator, are assigned y = 0.

The Cross-Entropy loss is found to be very common in the classification tasks (the loss function has also been used in the previous labs). Given a label y (either 0 or 1, for binary classification) and the predicted probability ŷ (a decimal number that ranges from 0 to 1).

For Real Image Y: Recall that, depending on the definition given before, the label for the actual image is "1". When we input, passing the actual image Y through the discriminator D, the results given by D(Y) show the extent to which the image Y is genuine. Next, we define the Cross-Entropy loss using the given data based on its structure. We have y=1 while ŷ=D(Y).

For Fake Image G(X): The label is "0". Here, one has the value "1-D(G(X))," wherein the parenthesis stands for the probability that, as a result, I = G(X) is an example of a fake image. Similarly, we have y = 0 for the phony image, and the probability that shows it is a fake image ŷ =1- D (G(X)).

In each step, the training batch includes real Y and the generated image G(X). After that, it summarizes the above two loss terms and helps the discriminator make correct predictions regarding real and fake photos. [4]

Since this is a binary classification problem, the loss function is identical to Binary Cross Entropy.

For Cross-Entropy (BCE) loss, we used real_label = 1 and fake_label = 0 for the PyTorch function. Then, the equation of BCE loss in PyTorch can be written as follows:

$$L = -w_n[y_n \cdot \log x_n + (1-y_n) \cdot \log(1-x_n)]$$

$$L_{Dreal} = -[1 \cdot \log x_n + (1-1) \cdot \log(1-x_n)]$$

$$L_{Dreal} = -[1 \cdot \log x_n]$$

$$L_{Dfake} = -[0 \cdot \log x_n + (1-0) \cdot \log(1-x_n)]$$

$$L_{Dfake} = -[1 \cdot \log(1 - x_n)]$$

Equation of BCEloss in pytorch (sigmoid):

$$L \ =-w_n[y_n \cdot \log x_n + (1-y_n) \cdot \log(1-x_n)]$$

BCE is used for "is it?" questions, such as LR output probability, the probability of raining or not raining tomorrow.

Equation of CEloss in pytorch (softmax):

$$L \ =-w_n[\log \frac{e^{x_{n,y_n}}}{\sum_{C=1}^{C} e^{x_{n,c}}}]$$

CE is used for "which is it" questions, such as multi-classification problems.

Thus, the generator envisages creating images of high quality that will effectively deceive the Discriminator. The training process can minimize the gap between the generated and the actual photos, taking advantage of the opinion of the Discriminator in improving the generation.

The output value D(G(X)) represents how likely the Discriminator believes the input image G(X) to be an actual face image. Ideally, the generator wants to look for a probability D(G(X)) that is as significant as possible. Therefore, for the generator, the label of the generated image G(X) is “1,” and the probability that the generated image is an actual image is ŷ=D(G(X)). Discriminator has the label “0”. Here is the Adversarial learning of the generator and the Discriminator.[5]

The training procedure in GAN consists of two stages: the Discriminator and the generator. The generator is engineered to generate images to deceive the Discriminator, while the Discriminator is trained to have a better capacity to distinguish between real and fake photos. This adversarial training goes on until the generated images merge with authentic images.

It is manifested in the cost estimates, where this adversarial approach of the two teams involved in the process is expressed. In other words, when training the Discriminator, it wants the probability D(G(X)) to be as small as possible and the label as “0” for the fake face image generated. When training the generator, it would love to have the probability of D(G(X)) as high as possible but with the label “1” so that the Discriminator will be tricked into giving the wrong prediction. These two networks work in opposition to each other while evolving in tandem.

For the backward propagation, we need to calculate the forward propagation for the batch. Suppose we input 64 images into the model, and forward propagation is done on all the photos simultaneously. Next, the prediction for all 64 images is obtained, so the loss function is used to calculate the loss function for the loss calculation.

$$\frac{1}{64}\sum_{i=1}^{64} Loss(y_i,\widehat{y_i})$$

(Where $y_i$ refers to the actual class and $\widehat{y_i}$ represents and estimated class for i-th image.)

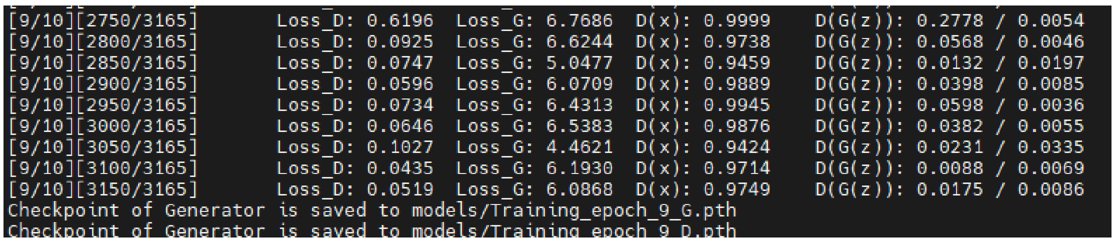
Then, we can use the total loss to do backward propagation if needed. In this step, two things are obtained: gradients of the loss concerning the model's parameters (weights). After calculating these gradients, the model weights are updated, usually performed by an optimization algorithm (like SGD, Adam, etc.). The update is carried out just once for the whole batch.

$$w = w - a\,\nabla L$$

(Where w means the weights, $a$ means the learning rate, and ∇L represents the average from the gradients of the loss functions calculated with the batch.)[6]

## Result

256*256 image generation result is shown below:



```
[9/10][2750/3165]       Loss_D: 0.6196  Loss_G: 6.7686  D(x): 0.9999    D(G(z)): 0.2778 / 0.0054
[9/10][2800/3165]       Loss_D: 0.0925  Loss_G: 6.6244  D(x): 0.9738    D(G(z)): 0.0568 / 0.0046
[9/10][2850/3165]       Loss_D: 0.0747  Loss_G: 5.0477  D(x): 0.9459    D(G(z)): 0.0132 / 0.0197
[9/10][2900/3165]       Loss_D: 0.0596  Loss_G: 6.0709  D(x): 0.9889    D(G(z)): 0.0398 / 0.0085
[9/10][2950/3165]       Loss_D: 0.0734  Loss_G: 6.4313  D(x): 0.9945    D(G(z)): 0.0598 / 0.0036
[9/10][3000/3165]       Loss_D: 0.0646  Loss_G: 6.5383  D(x): 0.9876    D(G(z)): 0.0382 / 0.0055
[9/10][3050/3165]       Loss_D: 0.1027  Loss_G: 4.4621  D(x): 0.9424    D(G(z)): 0.0231 / 0.0335
[9/10][3100/3165]       Loss_D: 0.0435  Loss_G: 6.1930  D(x): 0.9714    D(G(z)): 0.0088 / 0.0069
[9/10][3150/3165]       Loss_D: 0.0519  Loss_G: 6.0868  D(x): 0.9749    D(G(z)): 0.0175 / 0.0086
Checkpoint of Generator is saved to models/Training_epoch_9_G.pth
Checkpoint of Generator is saved to models/Training_epoch_9_D.pth
```

The following fields are:

(a) Total epochs divided by current epoch

(b) The epoch's current iteration divided by its total iterations

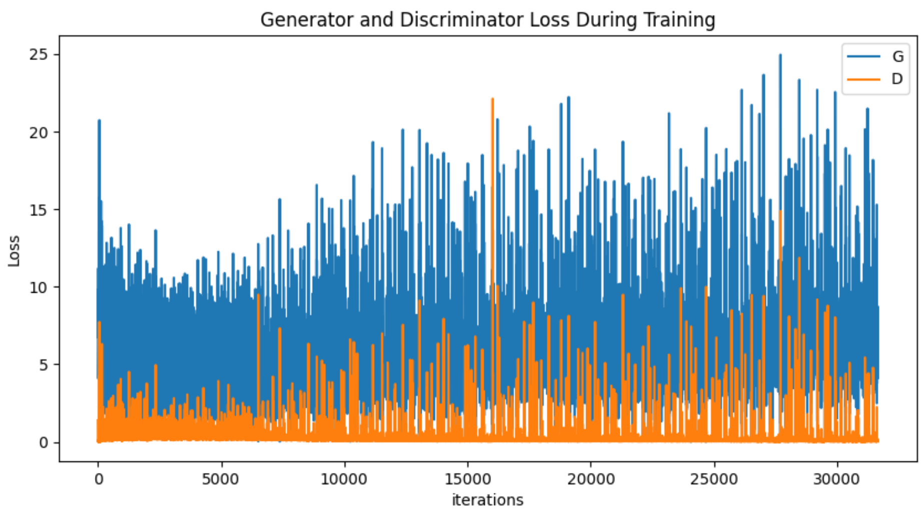
(c) The mean loss value during discriminator training

(d) The generator's training average loss value

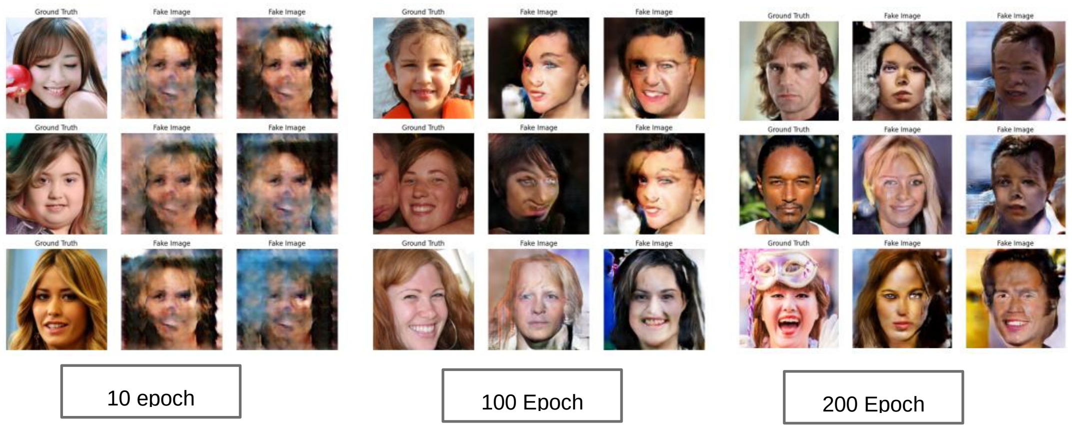
(e) When training the discriminator

(f), the real face images' categorization loss When training the discriminator or the generator, the categorization loses the generated face images.

I create a "train_loss.png" after the training is complete to illustrate how the loss has changed, as seen below:



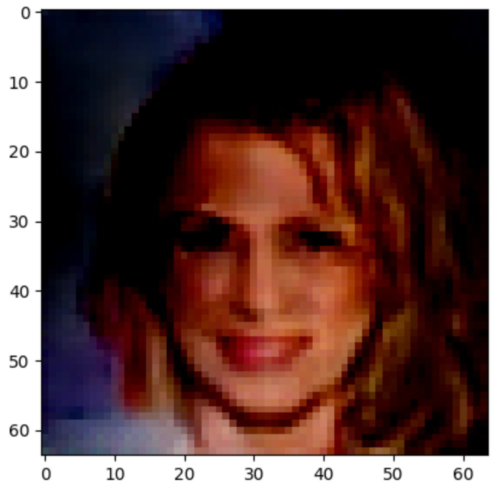
The file "result" contains a few intermediately generated face images for each epoch. During the training cycles, we may observe that the generated face images resemble real faces more.



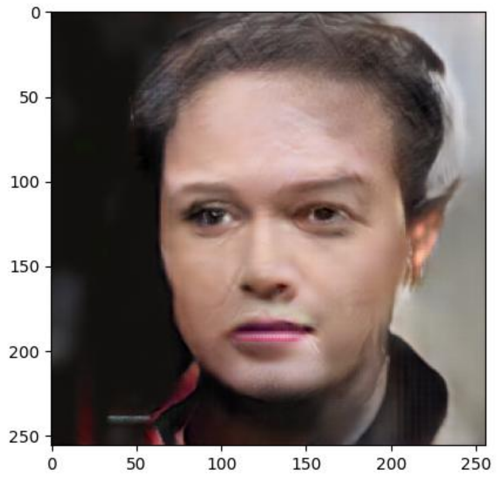
## Observation

Observation on GANS between one to ten epoch:
**Detail and Clarity:** Comparing epoch 100 and epoch 10, the images in epoch 100 seem to have a better resolution and contrast. This indicates that with model training epochs, the GAN has enhanced in producing more realistic facial structures and skin accounts.
**Variability:** It can be observed that the variability of the generated faces tends to increase by epoch 100. The variations in the expressions, the hair style and everything else on the face seems to be larger suggesting that the model is learning to capture more of the human face.

Remarks on Realizations with GANs of Different Image Size



GAN (64*64)



GAN (256*256)

Following I will explain the observation on GANs with Different Image Sizes from four dimensions: [7]

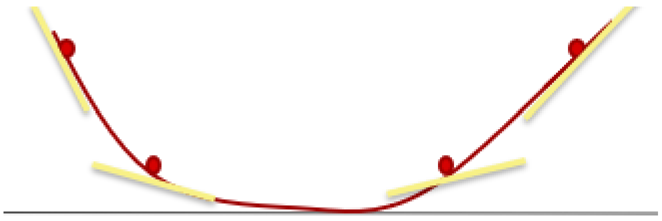
Feature Extraction:
**256x256 Images:** This is so because larger image sizes enable the GAN to retrieve higher levels of features from the image. This means details of features such as eyes, nose, and mouth, which are more accurately defined than the earlier representations.
**64x64 Images:** Lesser sizes of images may not capture them as effectively, and this results in features that look less detailed and sometimes blurred.
Generation of Unimportant Features:
**256x256 Images:** Though images with many pixels contain more detail, they produce extra low-relevance features. It can also, at the same time, create noise or characteristics that include unnecessary features within the image.
**64x64 Images:** The low resolution of the images provides good results since the GAN does not have the opportunity to overcrowd features with miscellaneous details.
Quality of Generated Images:
**256x256 Images:** The generated images generally have comparatively high quality when the needed sizes are significant; this is because the GAN learns to utilize more pixels to create the images.
**64x64 Images:** The images created here are in lower resolution, and the level of detail and realism in the high-resolution images is not captured here.

## Discussion

Why GAN can quickly generate the fake image in one epoch:

Iteration: Independently update the model's parameter based on the hypermeter (batch size). For example: if batch size is 100 & total image is 1000, it will need 10 iterations ($\frac{Total\ no.\ of\ images}{Batch\ size}$) to complete in one epoch. Therefore, 377 iterations mean that there are (64 * 377 = 24,128 images) used for training. [8]

It was found that optimized hyperparameters like Adam (Momentum + RMSProp) can help train networks properly. Latent noise is passed on to the image through the learned image.



Techniques that investigate generalization in GANs

In their study, Zhao et al. (2018), the authors employ systematically constructed training datasets to describe how existing models create new attributes and their interactions. Some of the conclusions they reach are the following. Generalization in GANs: In general, both GANs and VAEs generate a sample set with different levels of generalization depending on the training sample set. Probability Distribution: The learned distribution prefers the mean number of objects if modes are near each other, even if the mean from the training set was not learned.

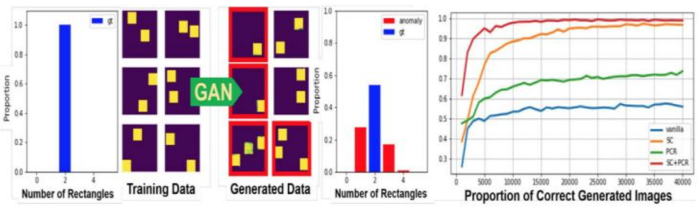


**Fig. 9.** Xuan et al. (2019) show that training a GAN over images with exactly two rectangles results in a model that generates one, two, or three rectangles (anomalous ones shown in red). They also propose a model that generates a high fraction of correct images (the right-most panel). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Alpha precision and recall

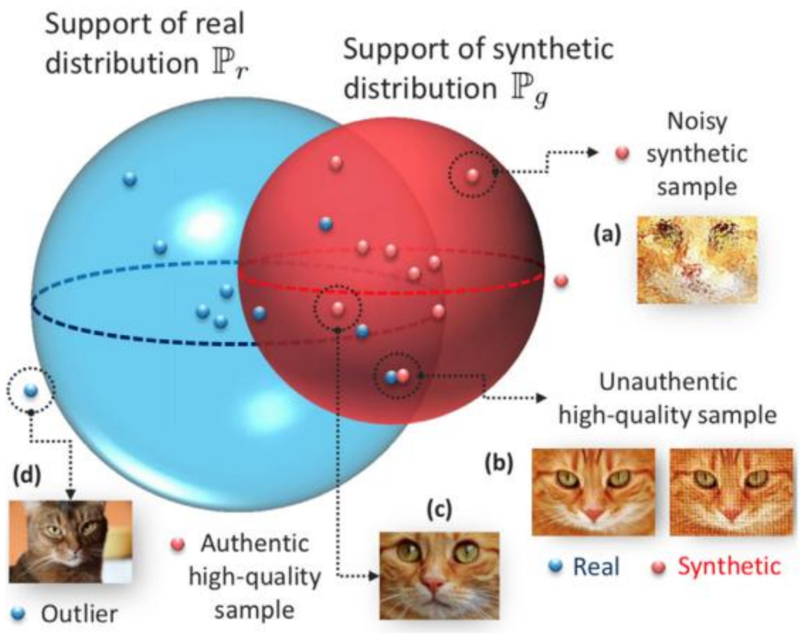
Alaa et al. (2021) describe a method for evaluating generative models, like GANs, using a 3-dimensional metric: α-Precision, β-Recall, and Authenticity.

**α-Precision:** Determining the number of the generated images that resemble the most typical or the most often occurring actual image. It aims at accuracy; that is, generated images should be as close as possible to the real ones, to the images most similar to real ones.

**β-Recall:** Measuring the extent to which synthetic samples' most typical β fraction includes actual samples. It stresses variety, which means that the model will reflect the sample data variety of the real world.

**Authenticity:** This metric measures the prospects that a synthetic sample provides a straight duplication of a training sample to minimize overfitting and enhance generalization. Check the model's ability to make new images that have never been seen before rather than taking screenshots of the photos it has been trained on.

**Hypersphere Embedding:** Data is inserted into hyperspheres where most samples are located at the core and outliers are at the periphery. Assists in determining which samples are most appropriate and which are deviant.[9]


Support of real distribution $\mathbb{P}_r$
Support of synthetic distribution $\mathbb{P}_g$
Noisy synthetic sample
(a)
Unauthentic high-quality sample
(b)
(c)
(d)
Authentic high-quality sample
Outlier
Real
Synthetic

Blue and red spheres represent $\alpha$ and $\beta$-supports of accurate and generative distributions,

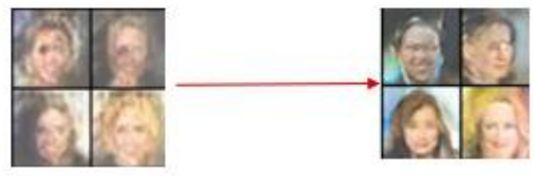
respectively. Blue points are actual data, while red points are synthetic data.
(a) Generated samples outside the blue sphere look unrealistic or noisy.
(b) Overfitted models can generate high-quality samples that are “unauthentic” because they are copied from training data.
(c) High-quality samples should reside in the blue sphere.
(d) Outliers do not count in the $\beta$-Recall metric. Here $\alpha = \beta = 0.9$, $\alpha$-Precision = 8/9, $\beta$-Recall = 4/9, Authenticity = 9/10.
Source: Figure from Alaa et al. (2021).

Why GANs Can Generate Desired Faces in this project
When trained, a GAN learns the specific features, patterns, and variations present in the dataset taken, in this case, human faces. The generator captures the statistical properties of the human face images, such as:

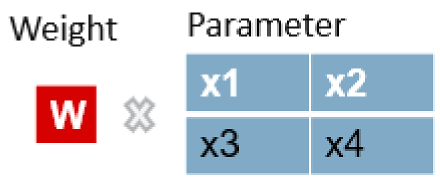
- Body parts (e.g., eyes, nose, mouth and ears)
- Differences in the orientation and positions, mouth)
- Skin tones and textures
- Hair styles and colors
- Variations in expressions and angles

In doing so, the generator is trained on various human faces so that it produces images similar to the training data during testing.[10]

Training Process:



**Step 1:** The generator will generate what is a false picture.
**Step 2:** The discriminator gives the image's outline of whether it is accurate or fake.
**Step 3:** The discriminator gives a feedback.
If the image is confirmed to be fake, the generator is adapted to generate more realistic pictures. That is why corrections are made to the discriminator for better results if the image is recognized as genuine.



In this case, we set real_label as 1, and fake_label as 0.
Equation of BCEloss in pytorch:

$$L \;= -w_n[y_n \cdot \log x_n + (1-y_n) \cdot \log(1-x_n)]$$

$$L_{Dreal} = -\,[1 \cdot \log x_n + (1-1) \cdot \log(1-x_n)]$$
$$L_{Dreal} = -\,[1 \cdot \log x_n\,]$$

$$L_{Dfake} = -\,[0 \cdot \log x_n + (1-0) \cdot \log(1-x_n)]$$
$$L_{Dfake} = -\,[1 \cdot \log(1-x_n)\,]$$

Training Discriminator:
Discriminator(real imgs) => 0.2333 => real face image
BCE(0.233, real label)
Equation: $-(\,1 \times \ln(0.233) + (1-1) \times \ln(1-0.2333)\,)$
= $-\big(\ln(0.233)\big) = 1.456$
Loss of real image = 1.456

Discriminator(fake imgs) => 0.1358 => fake noise image
BCE(0.1358, fake label)
Equation: $-(\,0 \times \ln(0.1358) + (1-0) \times \ln(1-0.1358)\,)$
= $-\big(\ln(0.8642)\big)$ = 0.1460
Loss of fake image = 0.1460

Loss of Discriminator = Loss of fake image + Loss of real image
= 1.456+0.1460 = 1.6013

Using the Loss of Discriminator to do Backpropagation to find the weights $\frac{dL}{dW}$.

After Training:
Discriminator(face image) = 1.0
Discriminator(fake image) = 0.0008
Meaning that Discriminator already can distinguish real and fake images.

Training Generator:
We want to fill the real label into discriminator to distinguish the fake image which is real image.
Discriminator(fake image) = 0.0008
BCE(0.0008, real label)
Equation: $-(\,1 \times \ln(0.0008) + (1-1) \times \ln(1-0.0008)\,)$
= $-(\,\ln(0.0008)\,)$ = 7.1128 => Adversarial learning

Using the Loss of Generator to do Backpropagation to find the weights $\frac{dL}{dW}$.

**Simultaneous Training:**

The training of the generator and the discriminator is done iteratively. Regarding the present objectives, the primary goal of the generator is to enhance its capacity to produce more believable visuals. The discriminator then hopes to improve its capacity to distinguish authentic images from fake ones.

**Convergence:** Gradually, the generator learns to create images that look as realistic as the human faces in real life do. The discriminator poorly recognizes the input images, including the generated and the real ones.
**Outcome:** The generator learns about images that are very realistic about human faces. The feedback from the discriminator facilitates the generation of a better output from the generator.[11]

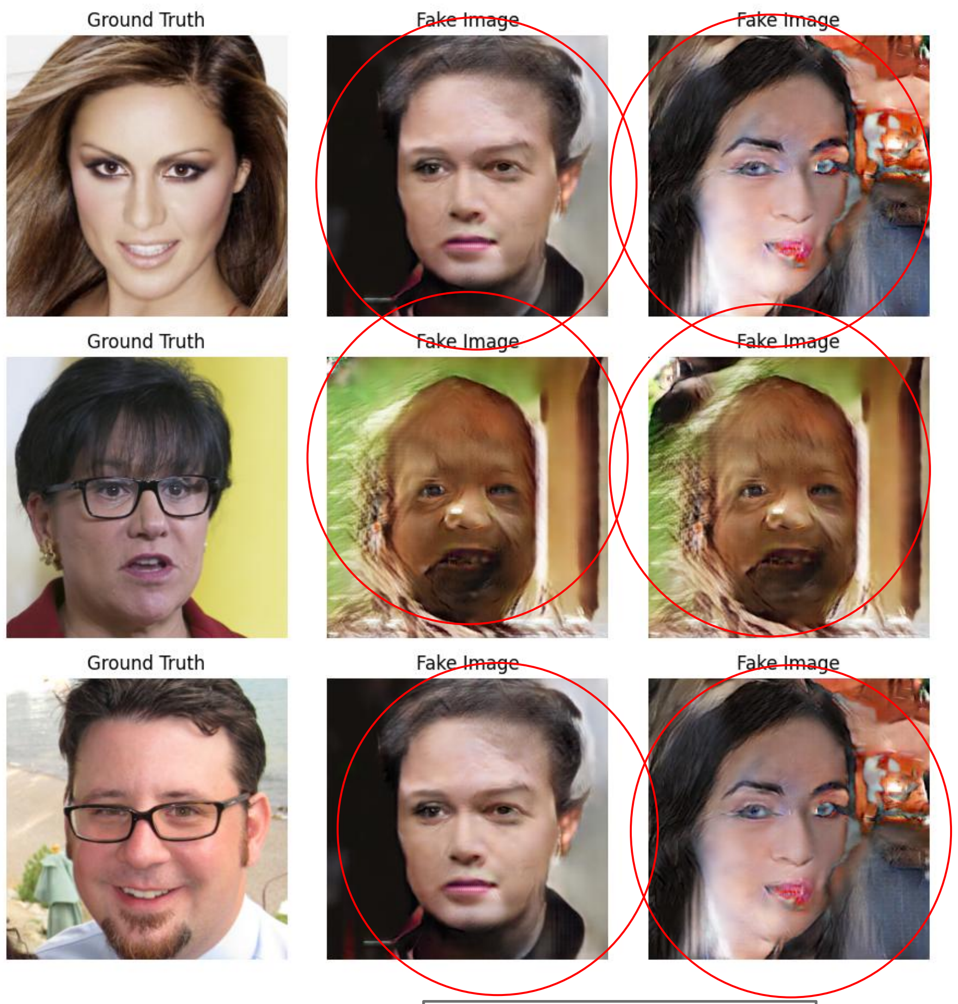
**Why Not Other Images?**
**Probability Distribution:** These learned probabilities probability distribution is a learned function of the training data. If the training set has only images of people's faces, then any images with dogs will be assigned low probability to those features.
**Feature Representation:** In this context, the model mimics the frequency distribution of predetermined features in the training set. About the generic nature of the breed, the GAN cannot express or create dog features as these are missing.

## Problem in my 256x256 GAN model

### Mode Collapse

The specific issue in Generative Adversarial Networks (GANs), known as mode collapse, results in generators producing restricted output variations (red circle) from diverse training datasets. The generator stops exploring all data distribution possibilities because it discovers several "modes" that successfully trick the discriminator. The generator in my PatchGAN experiment experienced mode collapse, leading to repetitive substandard images instead of generating detailed high-resolution outputs (e.g., 256x256 or beyond). The model failed to produce diverse textures and structures because it settled into generating blurry or repetitive patterns that did not capture the complexity of the target image. The problems you observed match the limited diversity and unclear details from GAN-based image generation tasks because of this fundamental challeng [12]

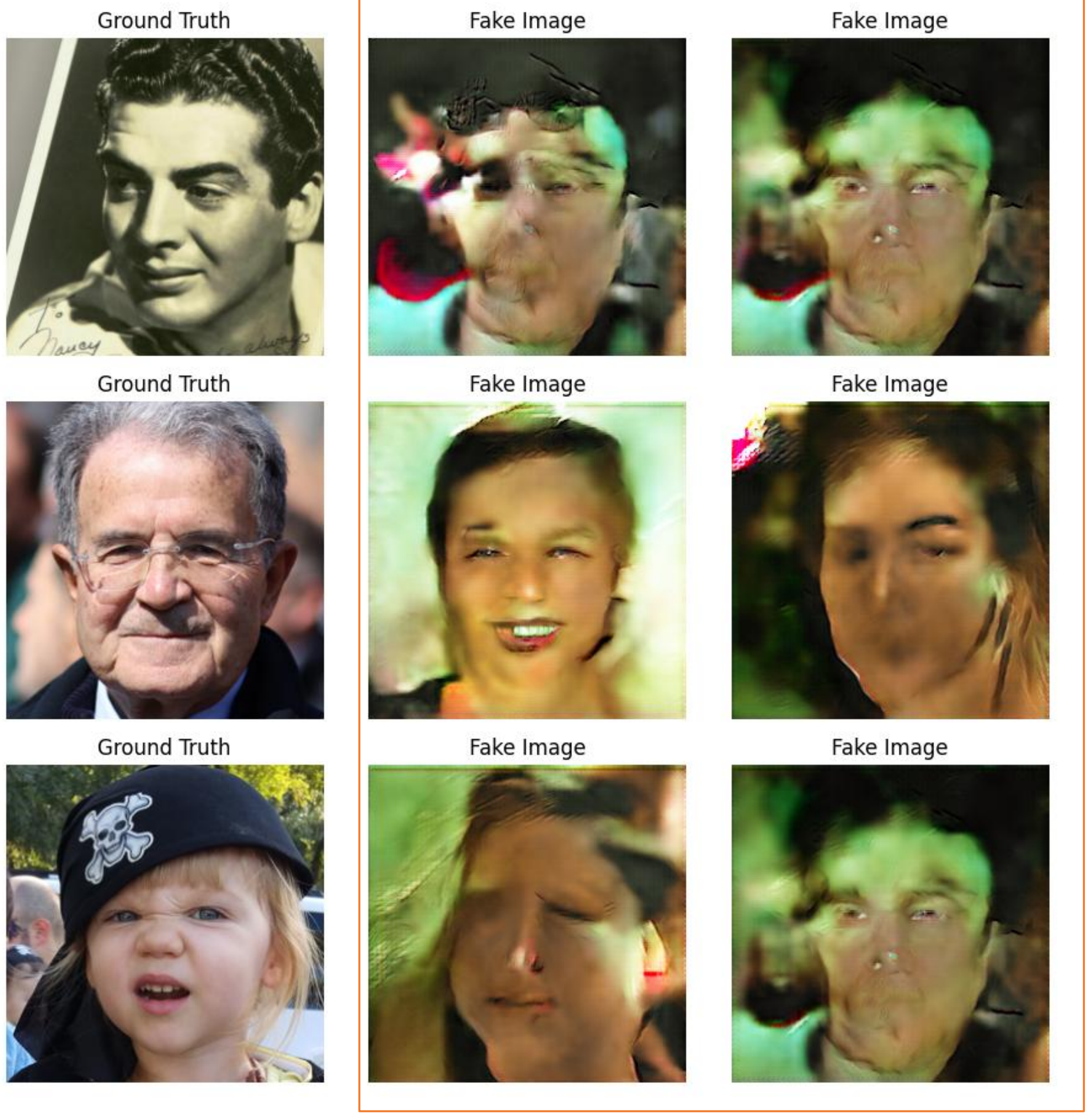
.



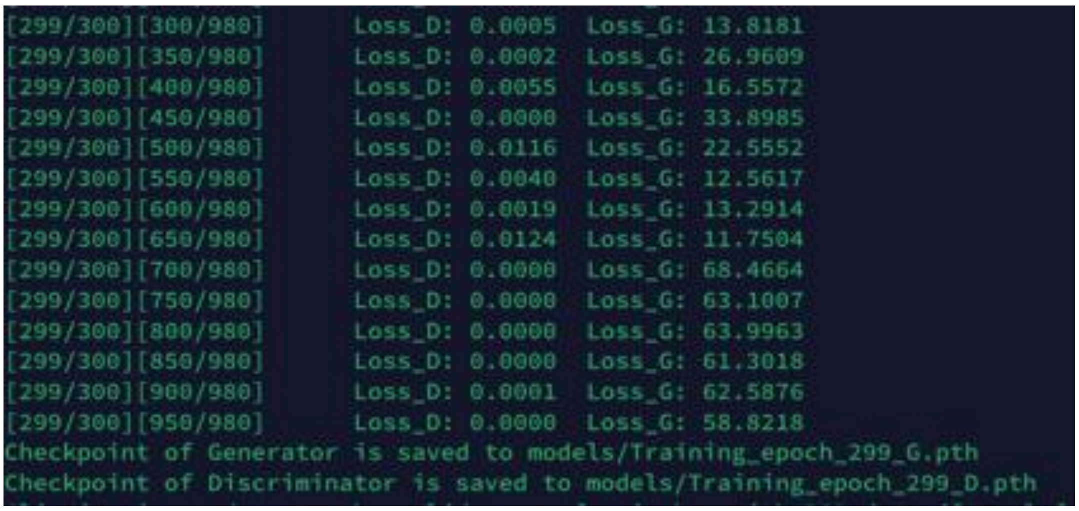
250 Epoch

**Overfitting**

The model develops overfitting when trained to such an extent that it recognizes all data specifics, including random variations, rather than applying learned patterns to new, unexpected entries. The model produces training sample duplicates while failing to create realistic or diverse outputs for new inputs when it overfits during image generation or upscaling tasks (e.g., CNN baseline for 64x64 to 256x256 images). When a CNN overfits, it tends to memorize training image patterns instead of developing robust output relations with resolution improvement. The model generates test data poorly because it learns training set-specific patterns, which produce images that lack diversity and show artifacts or distortions instead of precise generalized details. The possible overfitting behavior in my preliminary study could explain why my results came out blurry and low quality since the model failed to generalize with training examples.



300 Epoch



```
[299/300][300/980]        Loss_D: 0.0005  Loss_G: 13.8181
[299/300][350/980]        Loss_D: 0.0002  Loss_G: 26.9609
[299/300][400/980]        Loss_D: 0.0055  Loss_G: 16.5572
[299/300][450/980]        Loss_D: 0.0000  Loss_G: 33.8985
[299/300][500/980]        Loss_D: 0.0116  Loss_G: 22.5552
[299/300][550/980]        Loss_D: 0.0040  Loss_G: 12.5617
[299/300][600/980]        Loss_D: 0.0019  Loss_G: 13.2914
[299/300][650/980]        Loss_D: 0.0124  Loss_G: 11.7504
[299/300][700/980]        Loss_D: 0.0000  Loss_G: 68.4664
[299/300][750/980]        Loss_D: 0.0000  Loss_G: 63.1007
[299/300][800/980]        Loss_D: 0.0000  Loss_G: 63.9963
[299/300][850/980]        Loss_D: 0.0000  Loss_G: 61.3018
[299/300][900/980]        Loss_D: 0.0001  Loss_G: 62.5876
[299/300][950/980]        Loss_D: 0.0000  Loss_G: 58.8218
Checkpoint of Generator is saved to models/Training_epoch_299_G.pth
Checkpoint of Discriminator is saved to models/Training_epoch_299_D.pth
```

The need for improved architecture such as StyleGAN2 emerged because these challenges required solutions for generalization and diversity in high-resolution image generation.

## Reference

Acheampong, F. A., Nunoo-Mensah, H., & Chen, W. (2021). Transformer models for text-based emotion detection: A review of BERT-based approaches. *Artificial Intelligence Review, 54*(8), 5789–5829.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems, 27*, 2672–2680.

Göring, S., Rao, R. R. R., Merten, R., & Raake, A. (2023). Analysis of appeal for realistic AI-generated photos. *IEEE Access, 11*, 38999–39012.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local Nash equilibrium. *Advances in Neural Information Processing Systems, 30*.

Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., & Aila, T. (2020). Training generative adversarial networks with limited data. *Advances in Neural Information Processing Systems, 33*, 12104–12114.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097–1105.

LabML. (n.d.). Annotated deep learning paper implementations. GitHub. Retrieved February 11, 2025, from https://github.com/labmlai/annotated_deep_learning_paper_implementations

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. *Proceedings of the IEEE International Conference on Computer Vision*, 3730–3738.

Park, S. W., Kim, J. Y., Park, J., Jung, S. H., & Sim, C. B. (2023). How to train your pre-trained GAN models. *Applied Intelligence, 53*(22), 27001–27026.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Siu, W. C. (2021). *IEEE tutorial on deep learning*. Presented at IEEE Workshop, March 2021.

Sun, S., Luo, C., & Chen, J. (2017). A review of natural language processing techniques for opinion mining systems. *Information Fusion, 36*, 10–25.

Wang, W., Niu, L., Zhang, J., Yang, X., & Zhang, L. (2022). Dual-path image inpainting with auxiliary GAN inversion. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11421–11430.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2223–2232.