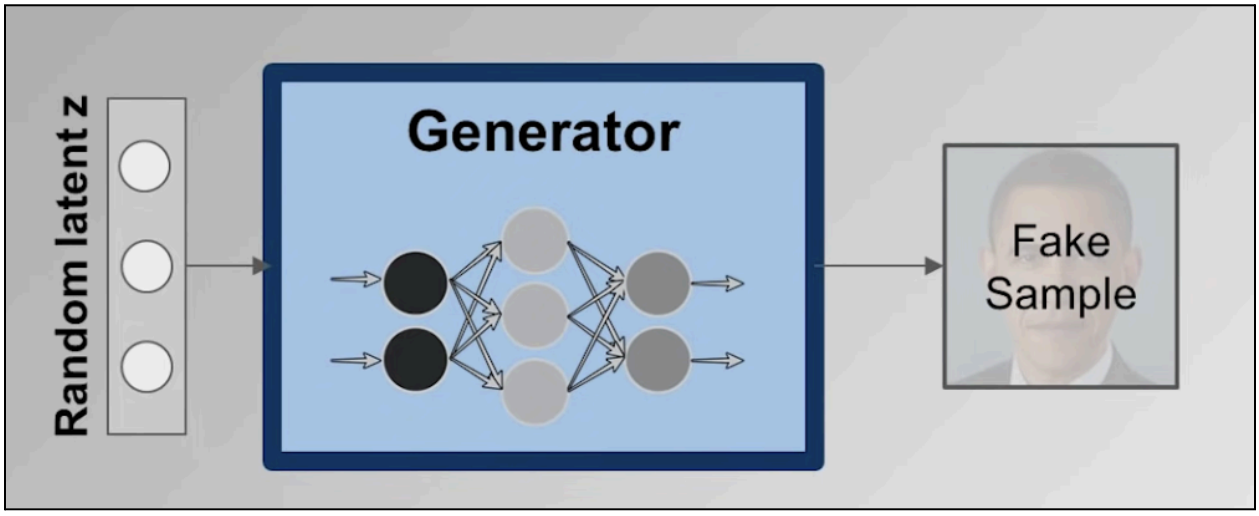
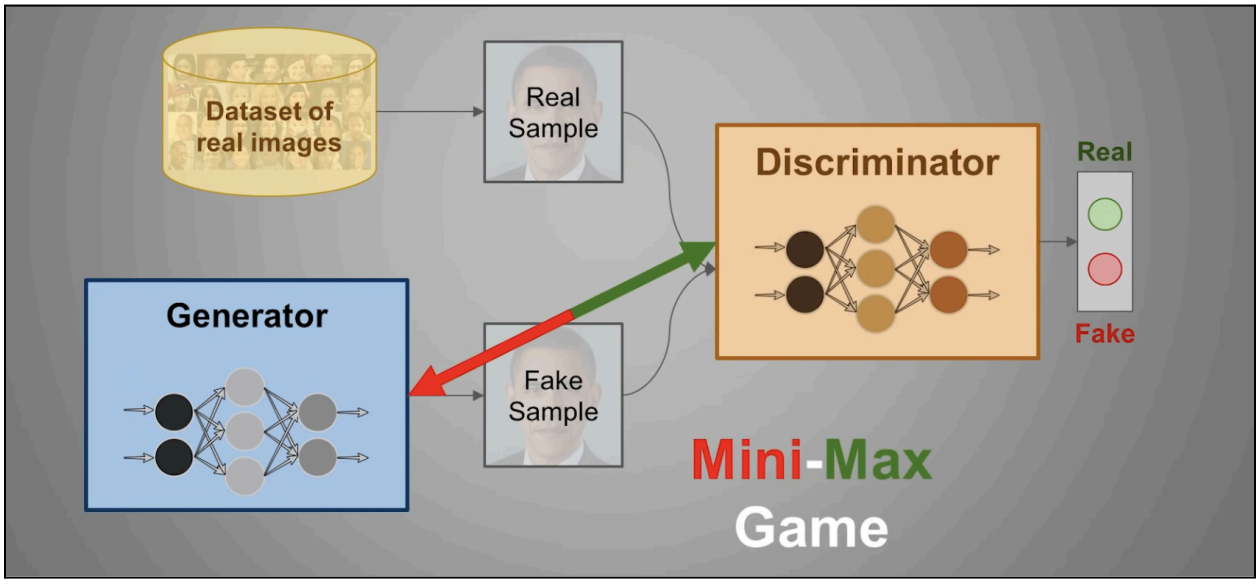
$$\min_G \; V(D,G) = -\log(4) + 2 \cdot JSD(p_{\text{data}} \| p_g)$$

0

$$\Rightarrow p_{data}(x) = p_g(x)$$



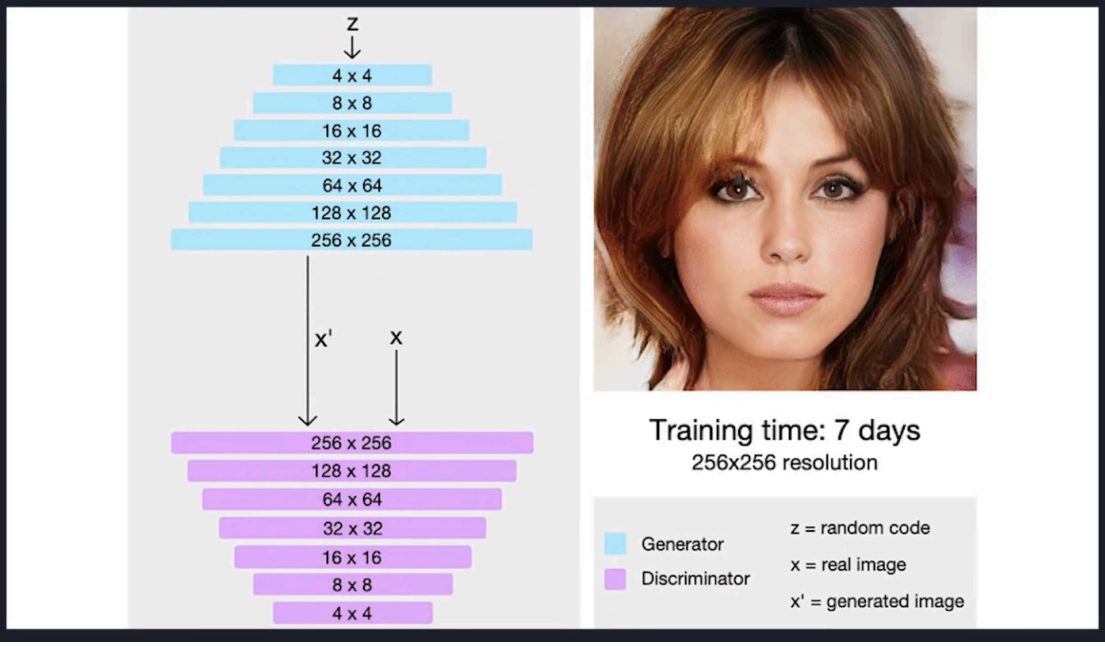
$$\min_G \max_D \mathbb{E}_{x \sim q_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

Training Loss

Mini-Max game:

- Minimize this loss wrt the Generator parameters
- Maximize this loss wrt the Discriminator parameters

# Progressive growing




8x8 then 16x16 then 32x32 then 64x64 then 128x128 then 256x256

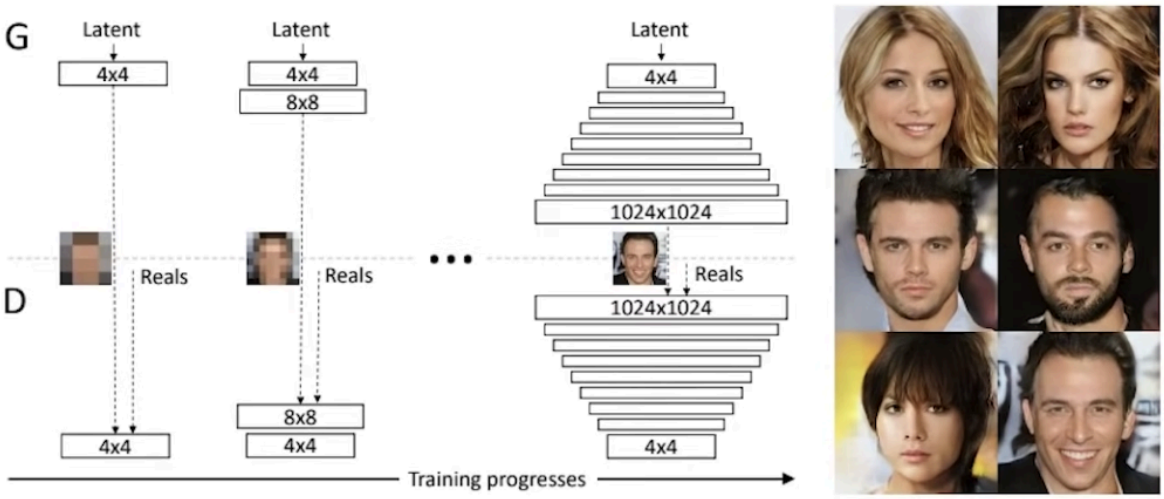
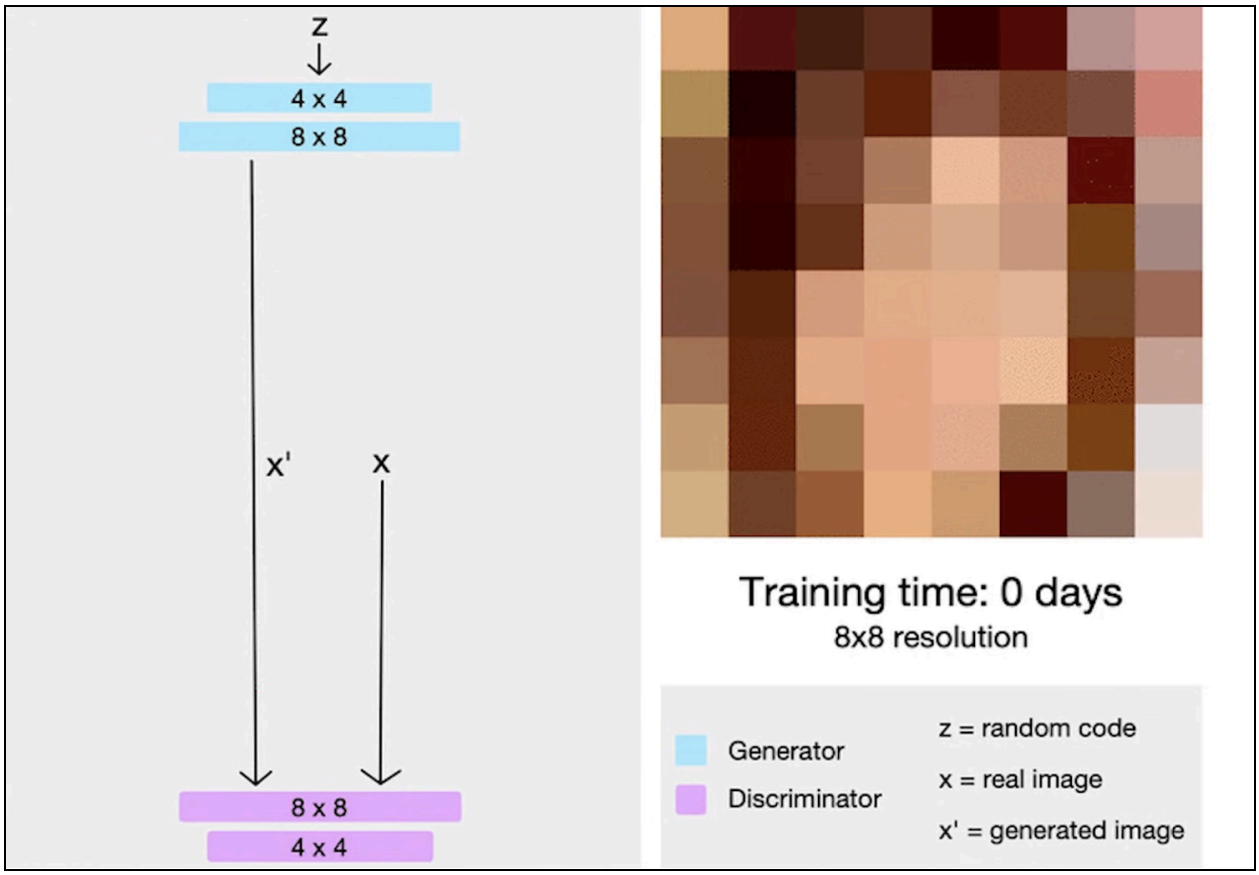
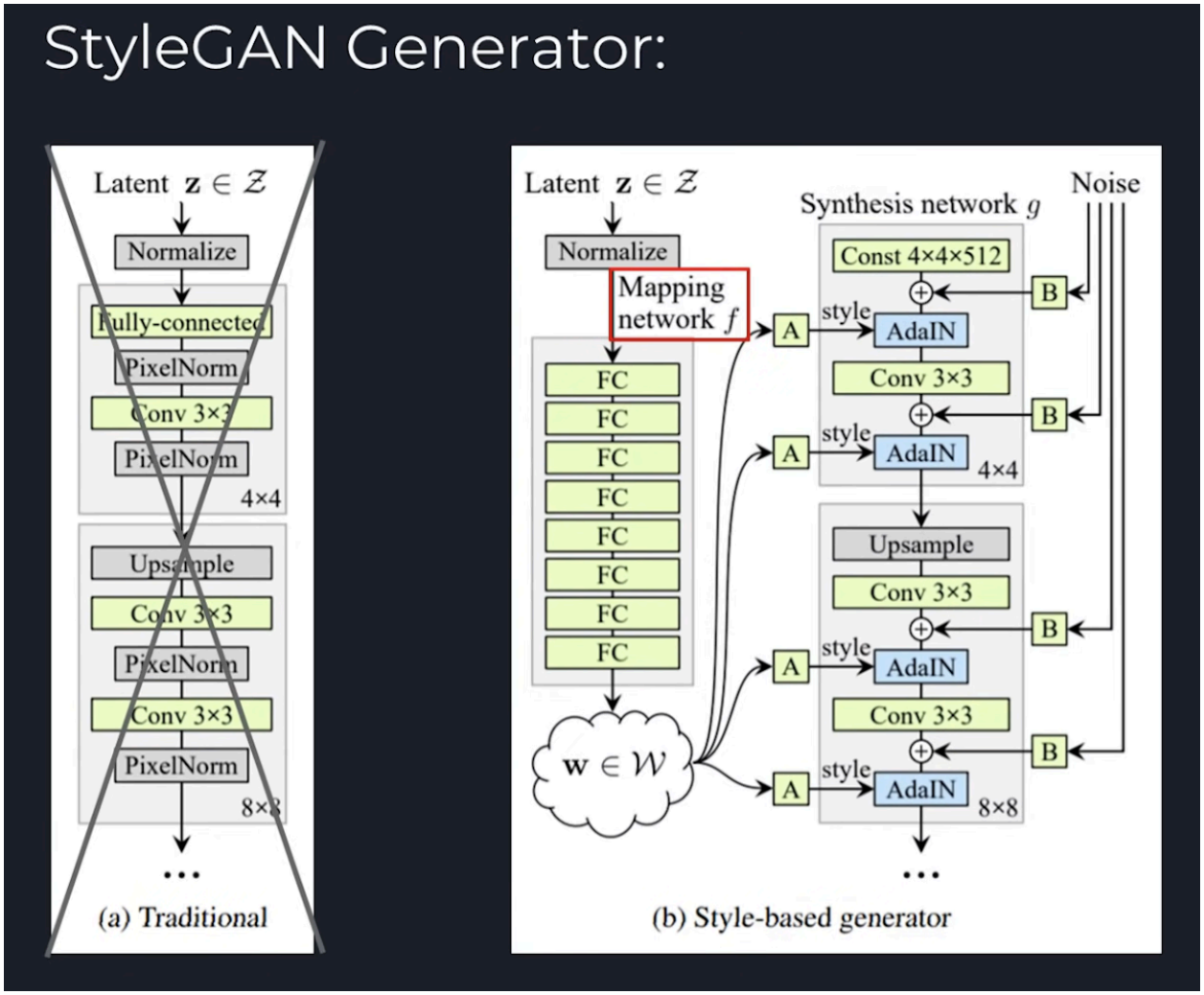



Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at $1024 \times 1024$.

first we are producing really small images and also the discriminator has to distinguish between really low resolution images, hence the network is quite stable and converges very quickly,,,,we then gradually go on to add more layers




what happens in real life is we go on increasing the layers during




1. W's distribution can be anything
2. Now the generator's input would a constant vector (a fixed seed ) and not a random noise like in progressive GANs.
3. AdaIN adds the output of the mapping layer into the architecture, (blending layer)
4. Here the upscalling can be correletated with how progressive gans work (train)
5. Noise is gaussian (B i mean) …why gaussian ? cause of smoothness? Cause of central limit theorem that the sum (or average) of many independent and identically distributed random variables tends toward a normal distribution, regardless of the original distribution of the variables
6. Why noise? To improve generalizability, as a regularization
7. What happens when we change where we add noise instead of adding in all layers?

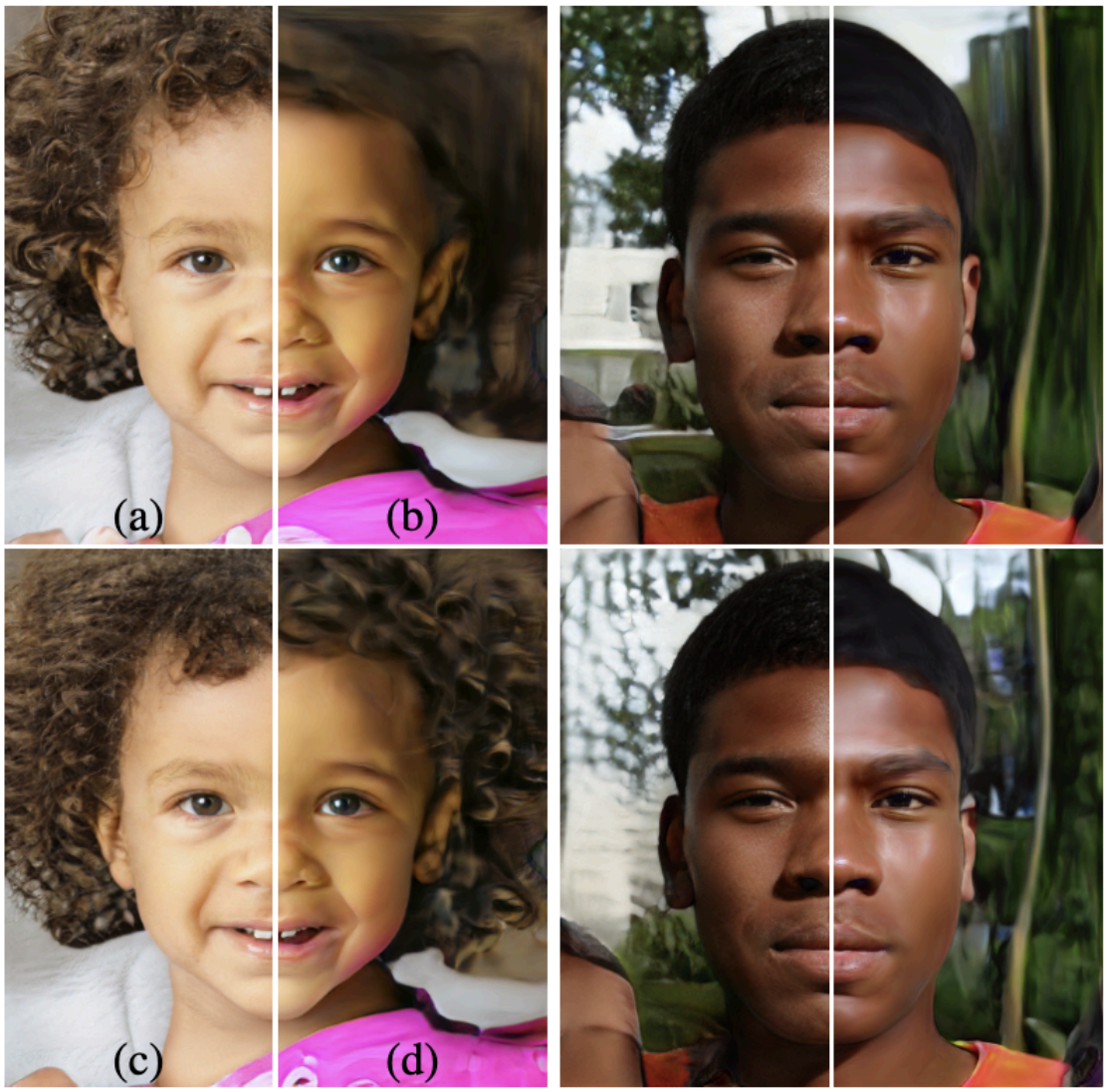



Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only ($64^2$ – $1024^2$). (d) Noise in coarse layers only ($4^2$ – $32^2$). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

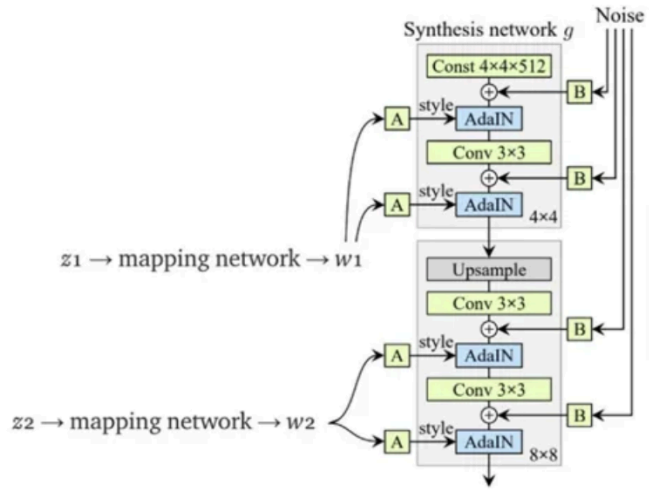
## 3.2. Stochastic variation

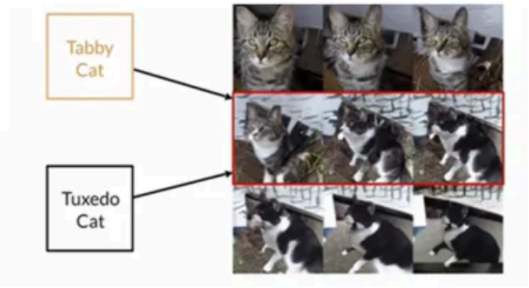
There are many aspects in human portraits that can be regarded as stochastic, such as the exact placement of hairs, stubble, freckles, or skin pores. Any of these can be randomized without affecting our perception of the image as long as they follow the correct distribution.

Fine → variations is less, as noise only affects the stochastic variables which are affected more in the coarse layers, but its proved (via empirical ideas mostly that this variation is localised due to the pressure the generator has (as talked bout in paper).

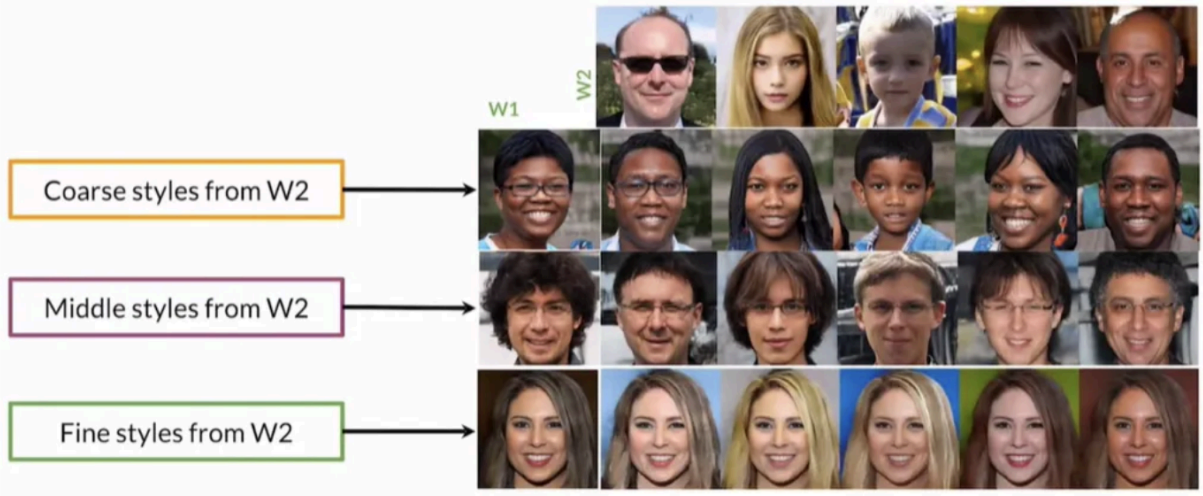
## Style Mixing & Mixing Regularization

'w' that was obtained from the noise mapping network is added to all the layers in the progressive growing block which gives us more control. Here we can mix multiple values of 'w' value such as w1, w2, w3 ... etc obtained from different 'z' values.



Since all the values can be controlled, we can obtain different images by adding different layers and one such of them is given below where w1 belongs to one style and w2 belongs to the other style. It also helps in producing a lot of diversity.



(a) Generated image (b) Stochastic variation (c) Standard deviation

Figure 4. Examples of stochastic variation. (a) Two generated images. (b) Zoom-in with different realizations of input noise. While the overall appearance is almost identical, individual hairs are placed very differently. (c) Standard deviation of each pixel over 100 different realizations, highlighting which parts of the images are affected by the noise. The main areas are the hair, silhouettes, and parts of background, but there is also interesting stochastic variation in the eye reflections. Global aspects such as identity and pose are unaffected by stochastic variation.

Figure 4 shows stochastic realizations of the same underlying image, produced using our generator with different noise realizations. We can see that the noise affects only the stochastic aspects, leaving the overall composition and high-level aspects such as identity intact. Figure 5 further illustrates the effect of applying stochastic variation to different subsets of layers. Since these effects are best seen in animation, please consult the accompanying video for a demonstration of how changing the noise input of one layer leads to stochastic variation at a matching scale.

We find it interesting that the effect of noise appears tightly localized in the network. We hypothesize that at any point in the generator, there is pressure to introduce new content as soon as possible, and the easiest way for our network to create stochastic variation is to rely on the noise provided. A fresh set of noise is available for every layer, and thus there is no incentive to generate the stochastic effects from earlier activations, leading to a localized effect.

Due to the absence of certain combinations in the training data, the mapping from the latent space (Z) to intermediate feature space (W) becomes curved. This curvature ensures that the missing or underrepresented combinations (e.g., long-haired males) are not easily reachable in the latent space Z. Essentially, it forces the model to generate intermediate features (W) that do not correspond to these invalid combinations, thus preventing the generation of unrealistic or improbable images.

## 2. Style-based generator

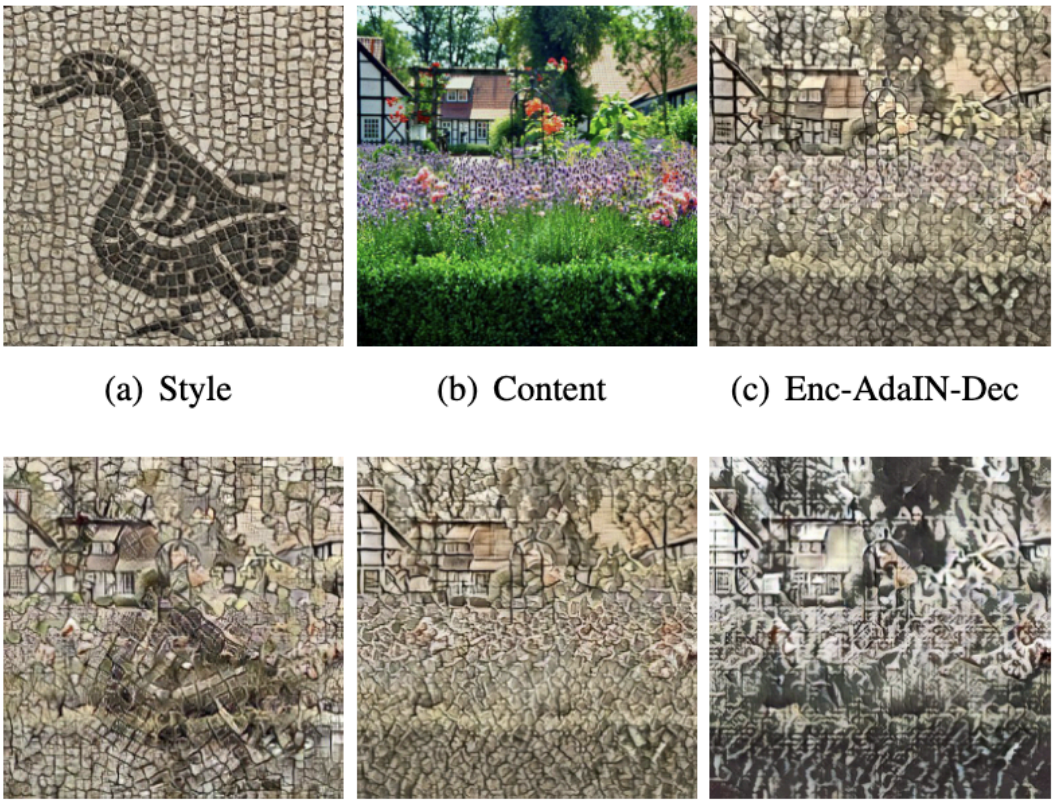
Traditionally the latent code is provided to the generator through an input layer, i.e., the first layer of a feed-forward network (Figure 1a). We depart from this design by omitting the input layer altogether and starting from a learned constant instead (Figure 1b, right). Given a latent code $\mathbf{z}$ in the input latent space $\mathcal{Z}$, a non-linear mapping network $f : \mathcal{Z} \to \mathcal{W}$ first produces $\mathbf{w} \in \mathcal{W}$ (Figure 1b, left). For simplicity, we set the dimensionality of both

Learned affine transformations then specialize $\mathbf{w}$ to *styles* $\mathbf{y} = (\mathbf{y}_s, \mathbf{y}_b)$ that control adaptive instance normalization (AdaIN) [27, 17, 21, 16] operations after each convolution layer of the synthesis network $g$. The AdaIN operation is defined as

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}, \qquad (1)$$

where each feature map $\mathbf{x}_i$ is normalized separately, and then scaled and biased using the corresponding scalar components from style $\mathbf{y}$. Thus the dimensionality of $\mathbf{y}$ is twice the number of feature maps on that layer.

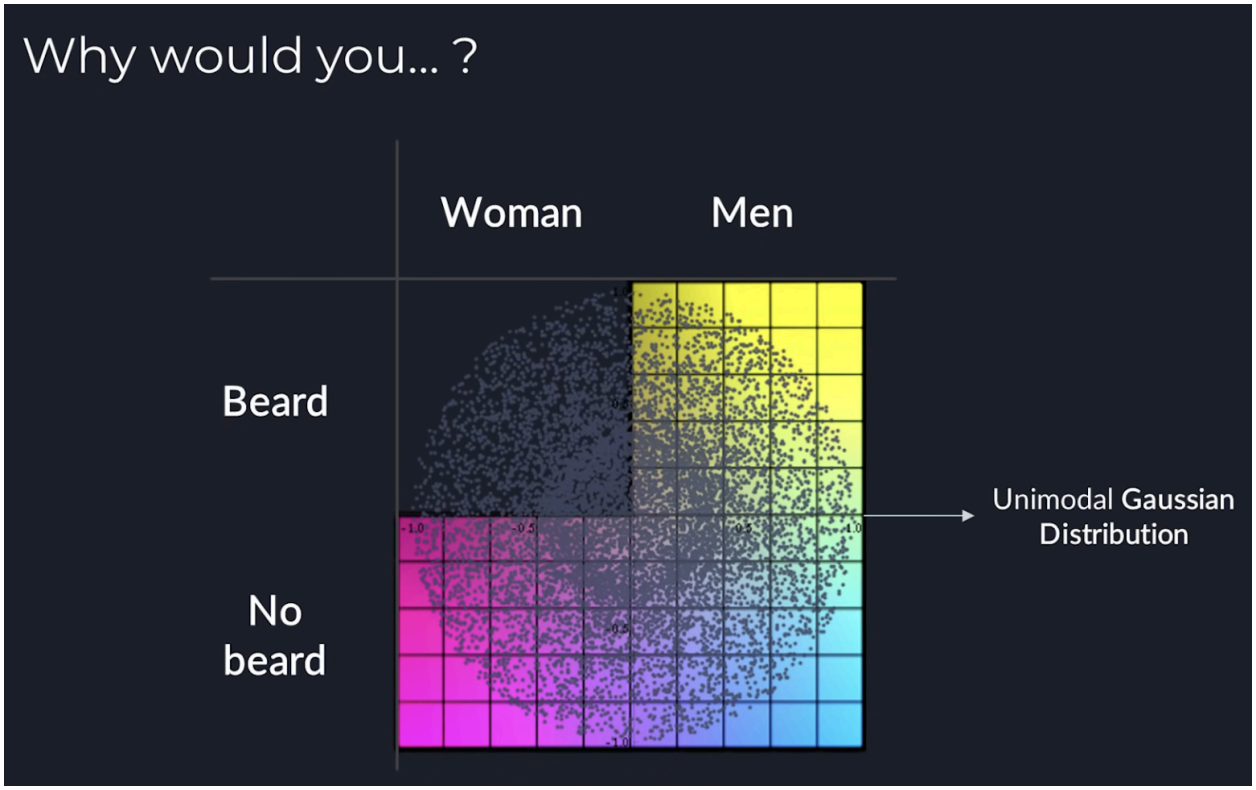
here we get the leverage to scale each feature map independently and biased to the scaling factor y (style) we try to mould the mean of xi towards y (as we want the style y, and how much we want for each instance can be changed)



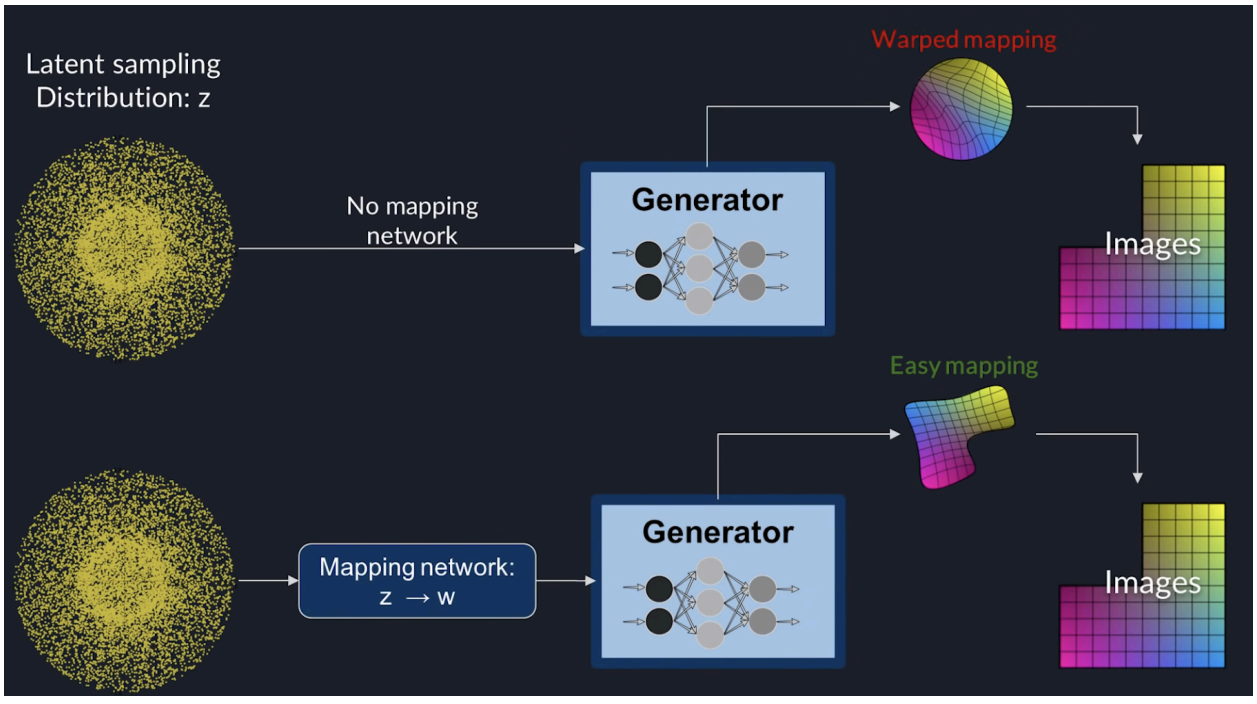
(a) Style (b) Content (c) Enc-AdaIN-Dec

(d) Enc-Concat-Dec (e) Enc-AdaIN-BNDec (f) Enc-AdaIN-INDec

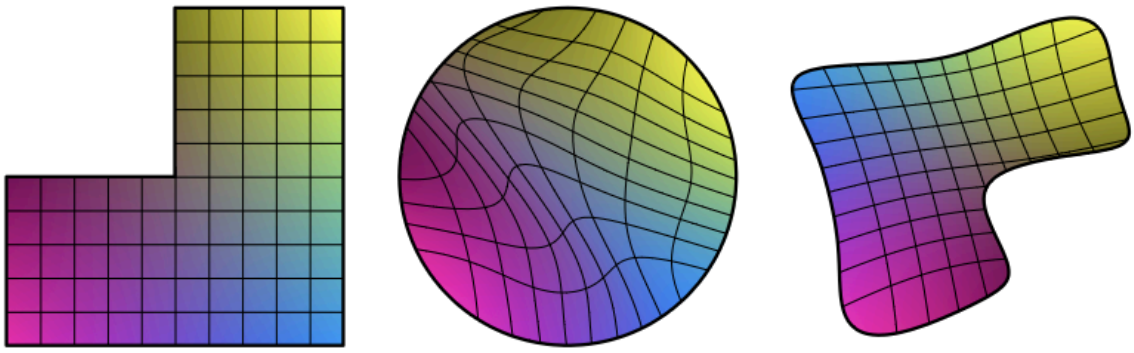
Figure 5. Comparison with baselines. AdaIN is much more effective than concatenation in fusing the content and style information. Also, it is important *not* to use BN or IN layers in the decoder.




Women with beard - where are you???, because of you we have a gap in the mapping.




idea is ki if we map the z vector by the mapping network itself then we would have an easy time mapping the generator's images onto the actual image distribution, as w is already in a good shape hence easy mapping (note w is a high dimensional space)

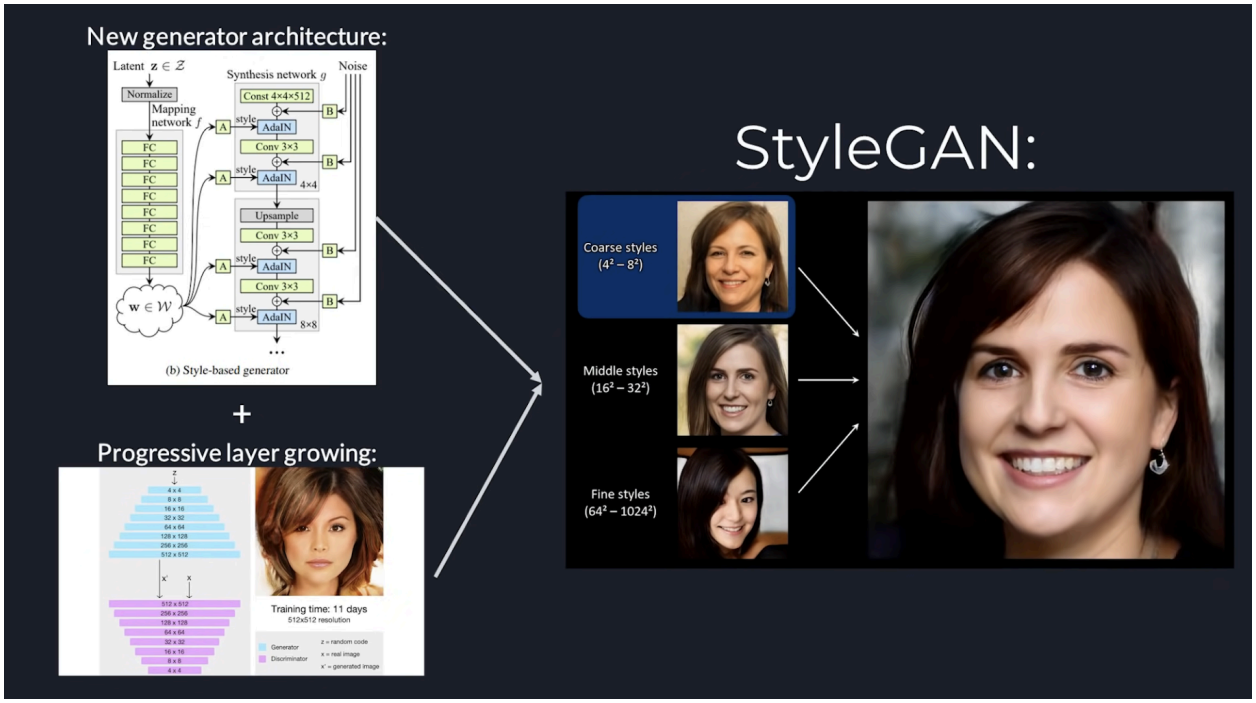


(a) Distribution of features in training set

(b) Mapping from $\mathcal{Z}$ to features

(c) Mapping from $\mathcal{W}$ to features

Figure 6. Illustrative example with two factors of variation (image features, e.g., masculinity and hair length). (a) An example training set where some combination (e.g., long haired males) is missing. (b) This forces the mapping from $\mathcal{Z}$ to image features to become curved so that the forbidden combination disappears in $\mathcal{Z}$ to prevent the sampling of invalid combinations. (c) The learned mapping from $\mathcal{Z}$ to $\mathcal{W}$ is able to "undo" much of the warping.



StyleGAN incorporates style mixing regularization during training, where a percentage of images are generated using two random latent codes instead of one. This encourages the network to learn more disentangled and localized representations of style attributes.