A. Implementation Details

Figure 7 presents all examples of style images which are used in computing relative distance and running the baseline on-the-fly stylization. In the subsections, we provide implementation details for DCGAN variants and StyleGAN2 variants. For further details, please refer to our code at https://github.com/naver-ai/FSMR.

A.1. DCGAN variants Experiments

Augmentations for DCGAN, bCRGAN include image flipping and random cropping. Consistency regularization coefficients for bCRGAN are $\lambda_{real} = \lambda_{fake} = 10$. We use non-saturating logistic loss with R_1 regularization, and Adam optimizer with $\beta_1 = 0.5$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and learning rate=0.0001 for both models. We ran our experiments on one Tesla V100 GPU, using Tensorflow 2.1.0, CUDA 10.1, and cuDNN 7.6.4. We apply FSMR for both real and fake samples and the total loss is

$$L_{\text{total}} = L_{\text{adv}} + \lambda L_{\text{FSMR}}, \tag{10}$$

where $\lambda = 10$. We performed all training runs using 1 GPU, continued the training for 20k iterations, and used a minibatch size of 32.

A.2. StyleGAN2 variants Experiments

For StyleGAN2, ADA[2], and DiffAug[3], we use the official Tensorflow implementations. We kept most of the details unchanged, including network architectures, weight demodulation, path length regularization, lazy regularization, style mixing regularization, bilinear filtering in all up-/downsampling layers, equalized learning rate for all trainable parameters, minibatch standard deviation layer at the end of the discriminator, exponential moving average of generator weights, non-saturating logistic loss with R_1 regularization, and Adam optimizer with $\beta_1 = 0$, $\beta_2 = 0.99$, and $\epsilon = 10^{-8}$. We ran our experiments on eight Tesla V100 GPUs, using Tensorflow 1.14.0, CUDA 10.0, and cuDNN 7.6.3. We apply FSMR only to the real samples because it leads to slightly larger gain. The weights λ for L_{FSMR} are 0.05 for FFHQ and 1 for the other datasets. We performed all training runs using 4 GPUs, continued the training for 25M iterations, and used minibatch size of 32, except for CIFAR-10, where we used 2 GPUs, 100M iterations, and a minibatch size of 64.

B. Evaluation metrics

We measure Fréchet Inception Distance (FID) [11] and Inception Score (IS) [33] using the official Inception v3 model in Tensorflow. When we compute FID and IS, we sample the same number of images to the number of real images in the training set.

[2] https://github.com/NVlabs/stylegan2-ada

[3] https://github.com/mit-han-lab/data-efficient-gans

Algorithm 2 FSMR Pseudocode, Tensorflow-like

```
# N: batch size, H: height, W: width, C: channels
def FSM(x, y, alpha, eps=1e-5):
    x_mu, x_var = tf.nn.moments(x, axes=[1,2],
        keepdims=True) # Nx1x1xC
    y_mu, y_var = tf.nn.moments(y, axes=[1,2],
        keepdims=True) # Nx1x1xC

    # normalize
    x_norm = (x - mu) / tf.sqrt(var + eps)

    # de-normalize
    x_fsm = x_norm * tf.sqrt(y_var + eps) + y_mu

    # combine
    x_mix = alpha * x + (1 - alpha) * x_fsm

    return x_mix # NxHxWxC

def discriminator(img, use_fsmr=False):
    # layers: conv, bn, actv, ..., fc ->
        discriminator layers

    x = img # NxHxWxC
    indices = tf.range(tf.shape(x)[0])
    shuffle_indices = tf.random.shuffle(indices)
    alpha = tf.random.uniform(shape=[], minval=0.0,
        maxval=1.0)

    for layer in layers:
       x = layer(x)
       if use_fsmr and layer.name == 'conv':
          y = tf.gather(x, shuffle_indices)
          x = FSM(x, y, alpha)

    return x # Nx1

def FSMR(real_img, fake_img, use_fsmr=True):
    real_logit = discriminator(real_img) # Nx1
    fake_logit = discriminator(fake_img) # Nx1

    if use_fsmr:
       real_logit_mix = discriminator(real_img,
           use_fsmr) # Nx1
       fake_logit_mix = discriminator(fake_img,
           use_fsmr) # Nx1
       d_fsmr_loss = l2_loss(real_logit,
           real_logit_mix)
       d_fsmr_loss += l2_loss(fake_logit,
           fake_logit_mix) # optional
       d_fsmr_loss *= 10 # weight for fsmr
    else:
       d_fsmr_loss = 0

    return d_fsmr_loss
```

C. Pseudo-code

We provide the Tensorflow-like pseudo-code of FSMR in Algorithm 2. FSMR is simple to fit easily into any model.

D. Comparison with previous mixing methods.

In Table 5, we show the comparison results from the previous mixing methods. First of all, the difference between the previous methods and FSMR is as follows. *CutMix* [38] and *Mixup* [39] apply augmentations on images, not on the feature maps. *Manifold-Mixup* [36] performs linear inter-

polation on the feature map without implicit style transfer. *StyleMix* [12] applies AdaIN on images with an extra encoder for augmentation, not for consistency. *MoEx* [27] shares only the first component with ours: reducing style bias requires not only feature statistics mixing (e.g. AdaIN [13], MoEx), but also the consistency loss after feature statistics mixing. When we have conducted the experiment, CutMix and Manifold-Mixup are applied as follows: CutMix in feature maps and Manifold-Mixup with consistency regularization. None of them outperforms ours because they do not reduce style bias.

	CIFAR-10	FFHQ	AFHQ
ADA w/ FSMR	**2.90**	**3.91**	**6.12**
ADA w/ Mixup	3.48	4.40	6.67
ADA w/ CutMix	3.45	4.36	6.51

Table 5. **Comparison from the previous mixing methods.**

E. Ablation on the style dataset.

Table 6 shows that the improvements by FSMR is larger than the improvements by using internal images as style references. The numbers (a) / (b) / (c) report FIDs for

(a) on-the-fly with the training images
(b) on-the-fly with WikiArt
(c) FSMR.

FSMR outperforms both on-the-fly settings. Hence, we argue that the difference does not come from using the same data distribution but from FSMR.

	CIFAR-10	FFHQ	AFHQ
DCGAN	16.02 / 15.88 / **14.98**	7.52 / 7.33 / **6.76**	14.87 / 14.22 / **13.19**
bCRGAN	12.58 / 12.43 / **11.17**	5.74 / 5.20 / **4.68**	9.02 / 8.63 / **8.33**

Table 6. **Comparison to on-the-fly stylization with the training images.**

F. Additional results

In Figure 8, we show the relative distance on several datasets except the ones in the main paper. We observe that FSMR reduces the relative distance of the discriminator, *i.e.*, the discriminator relies less on style.

Figure 9 illustrate an additional visualization of the effect of FSM on FFHQ, AFHQ, and LSUN Church.

In Figure 10, and 11, we show generated images for several datasets from the baseline with FSMR. The images were selected at random, *i.e.*, we did not perform any cherry-picking. We observe that FSMR yields excellent results in all cases.

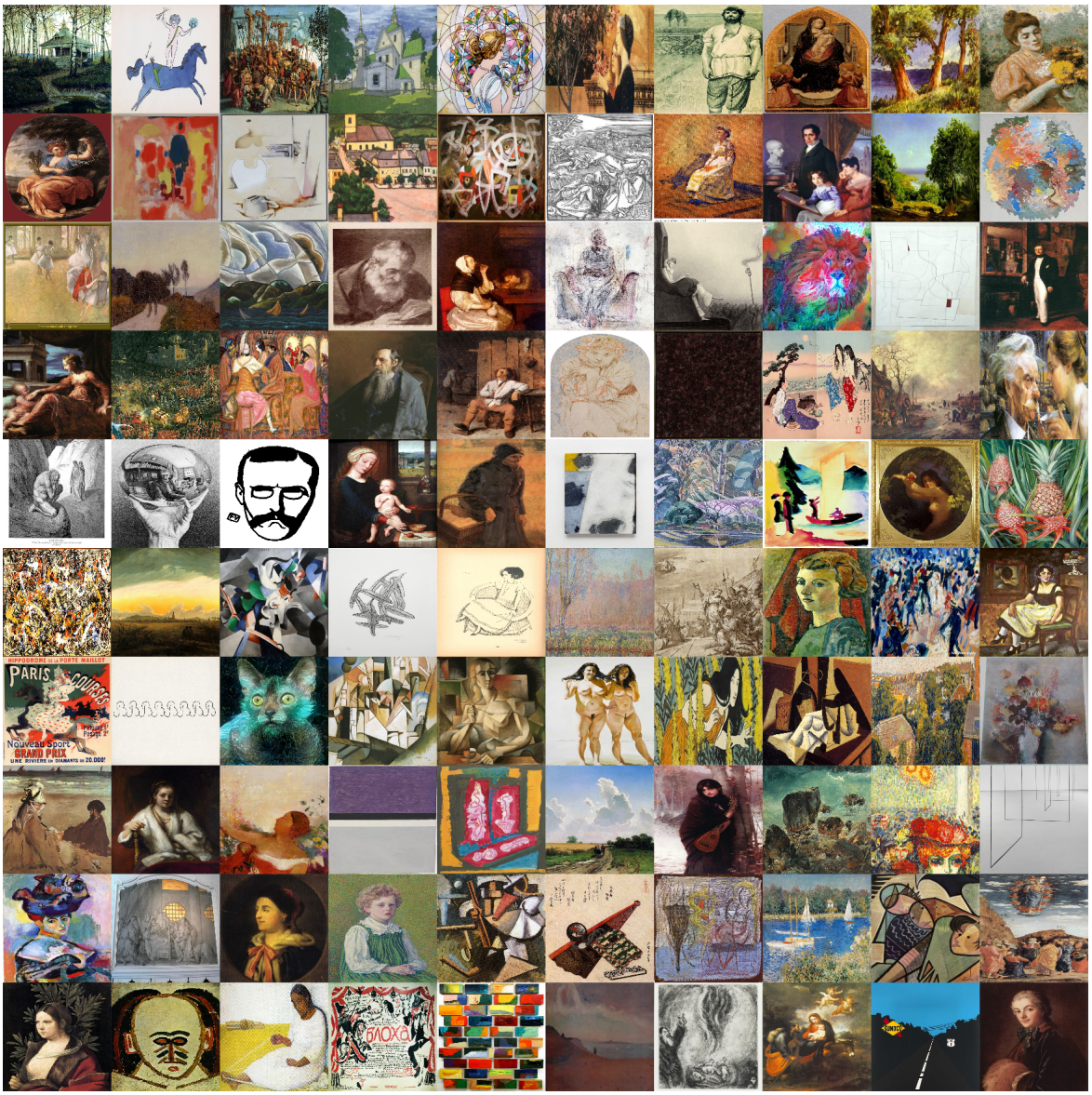


Figure 7. **Example style images**. We present all examples of style images that we used in the comparison experiments.

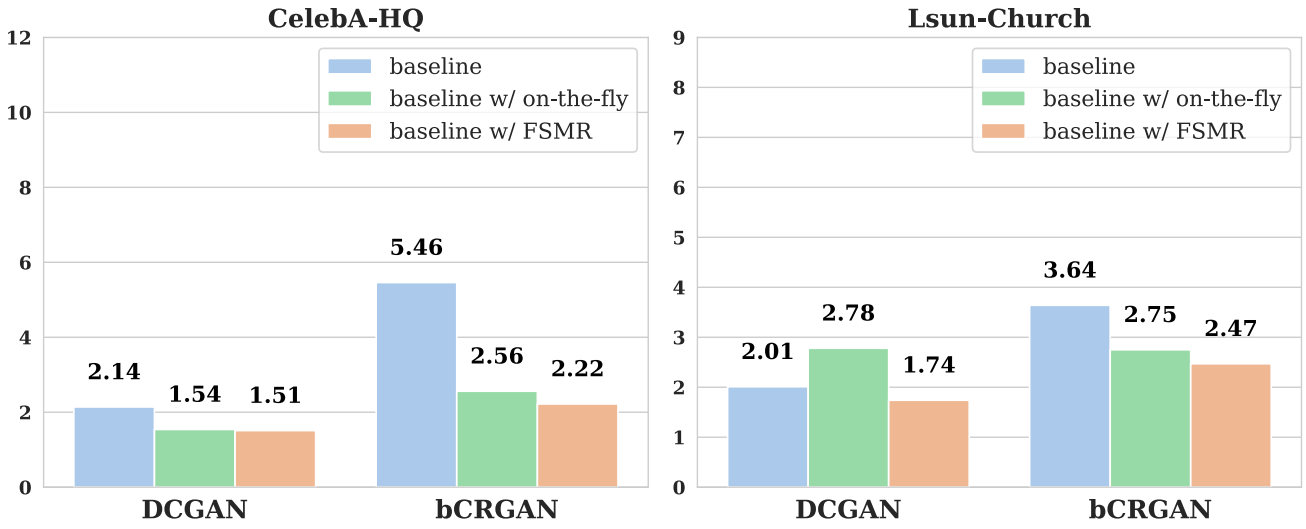
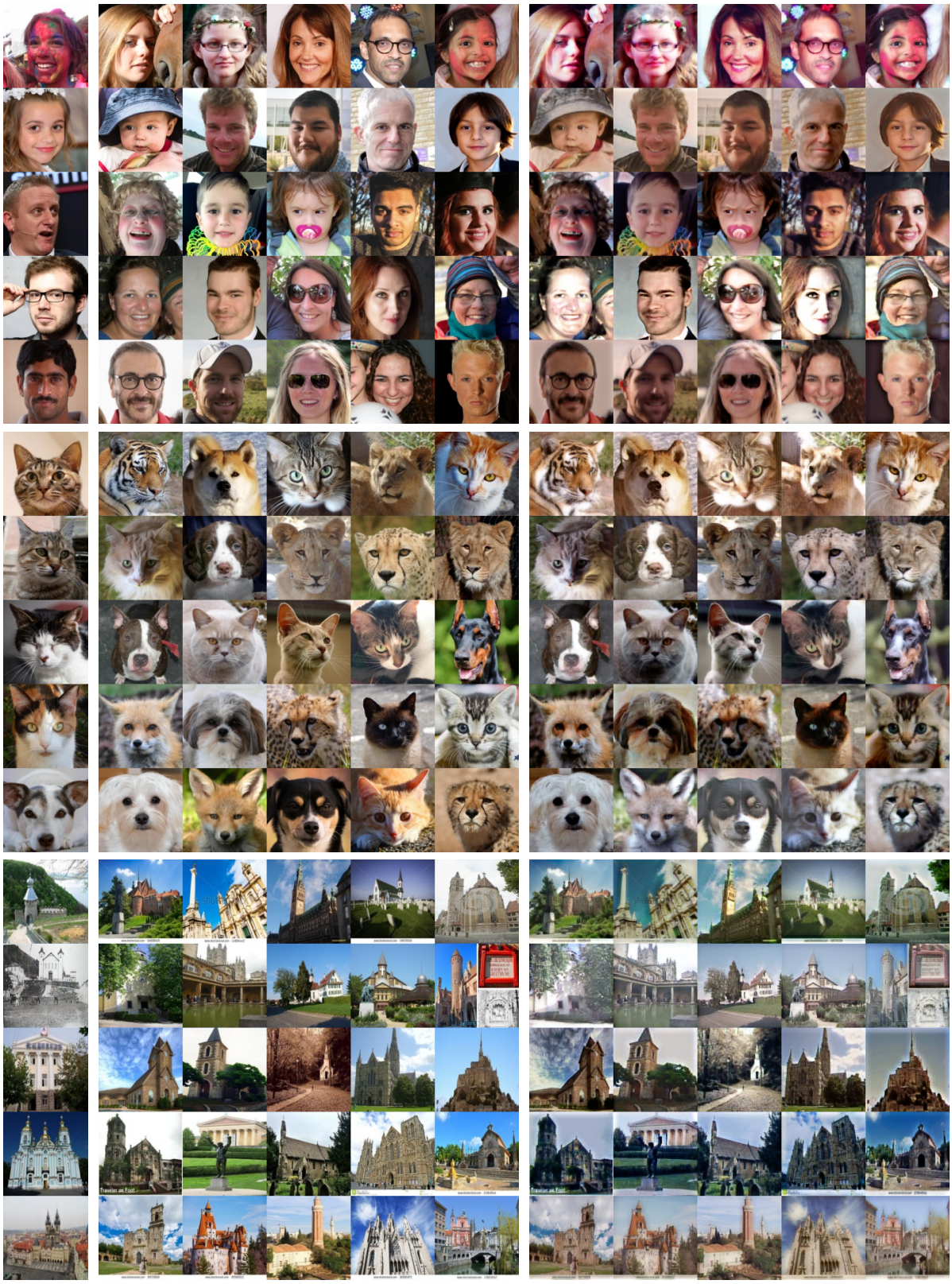


Figure 8. **The relative distance of discriminator** results for several datasets.



(a) Style (b) Content (d) Visualization of FSM

Figure 9. **Additional visualization of the effect of FSM**. (a) Style images. (b) Content images. (c) Reconstruction of FSMed features preserves the detailed shapes.

Figure 10. Some example images and FIDs of competitors on CIFAR-10, FFHQ, AFHQ, and MetFaces.

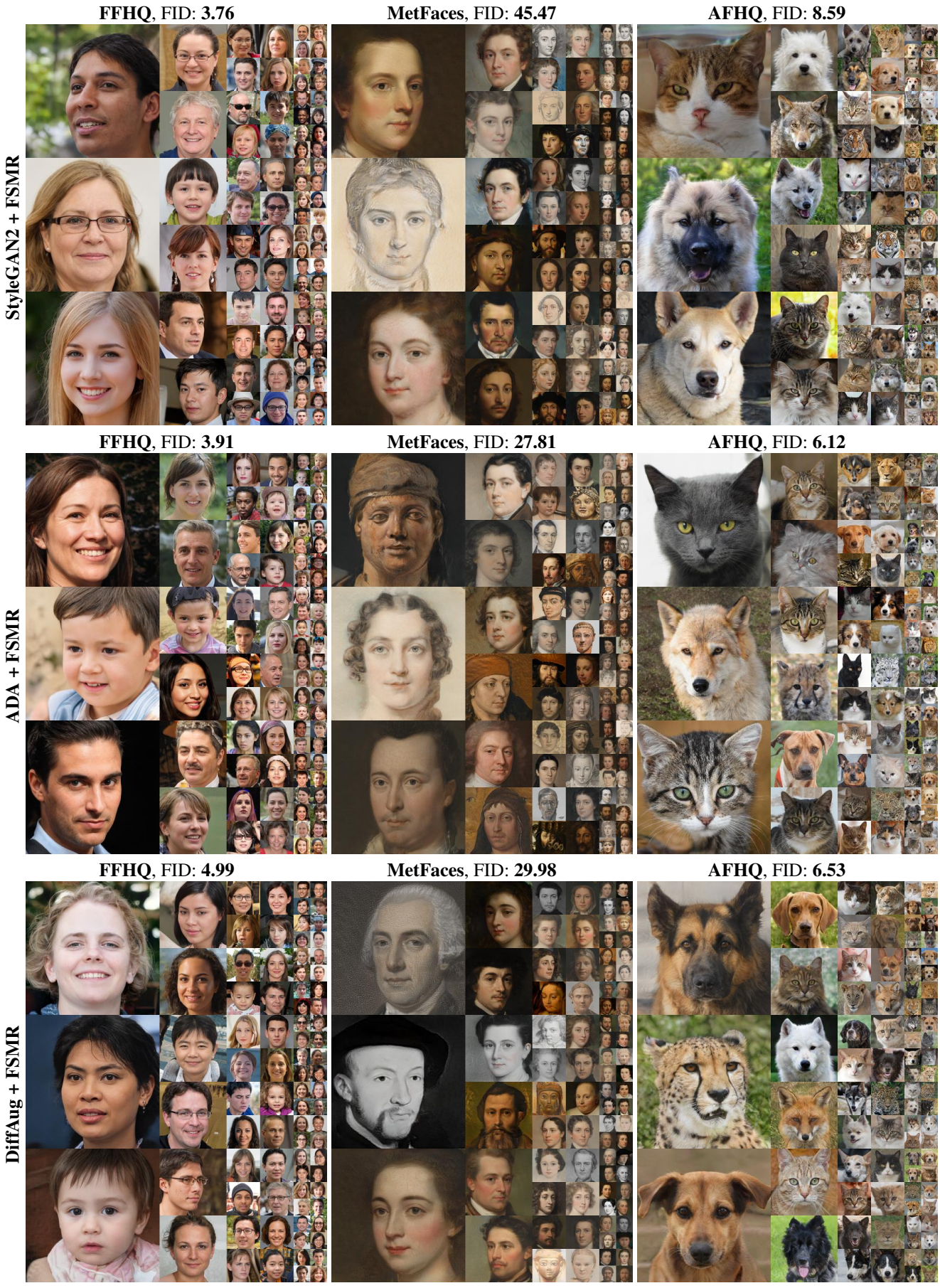


Figure 11. Uncurated random samples from the models trained with FSMR in three datasets.