Supplementary Material

ACTIVA: realistic single-cell RNA-seq generation with automatic cell-type identification using introspective variational autoencoders

Heydari et al.

Supplementary Material A : Complete Background

For convenience of the reader and ease of exposition, here we re-iterate the formulation of GANs and VAEs from Section 2 in the main text.

A.1. Generative Adversarial Networks

GANs (Goodfellow *et al.*, 2014) are capable of generating realistic synthetic data, and have been successfully applied to a wide range of machine learning tasks (Dziugaite *et al.*, 2015; Zhu *et al.*, 2016; Fedus *et al.*, 2018; Engel *et al.*, 2019) and bioinformatics (Liu *et al.*, 2019; Marouf *et al.*, 2020). GANs consist of a generator network (G) and a discriminator network (D) that train adversarially, which enables them to produce high-quality fake samples. During training, D learns the difference between real and synthetic samples, while G produces fake data to "fool" D. More specifically, G produces a distribution of generated samples P_g, given an input $z \sim P_z$, with P_z being a random noise distribution. The objective of GANs is to learn P_g, ideally finding a close approximation to the real data distribution P_r, so that $P_g \approx P_r$. To learn the approximation to P_g, GANs play a "min-max game" of

$$\min_G \max_D \mathbb{E}_{x\sim P_r} \log[D(x)] + \mathbb{E}_{z\sim P_z} \log[1 - D(G(z))],$$

where both players (G and D) attempt to maximize their own payoff. This adversarial training is critical in GANs' ability to generate realistic samples. Compared to other generative models, GANs' main advantages are (i) the ability to produce any type of probability density, (ii) no prior assumptions for training the generator network, and (iii) no restrictions on the size of the latent space.

Despite these advantages, GANs are notoriously hard to train since it is highly non-trivial for G and D to achieve Nash equilibrium (Wang *et al.*, 2019). Another disadvantage of GANs are vanishing gradients where an optimal D cannot provide enough information for G to learn and make progress, as shown by Arjovsky and Bottou (2017). Another issue with GANs is "mode collapse", that is, when G has learned to map several noise vectors z to the same output that D classifies as real data. In this scenario, G is over-optimized, and the generated samples lack diversity. Quantifying how well GANs have learned the distribution of real data is often complicated, consisting of measuring the dissimilarity between P_g and P_r when P_r is not known or assumed. Therefore, common ways of evaluating GANs involve direct evaluation of the output (Larsen *et al.*, 2016), which can be arduous.

Although some variations of GANs have been proposed to alleviate vanishing gradients and mode collapse (e.g. Wasserstein-GANs (WGANs) (Arjovsky *et al.*, 2017) and Unrolled-GANs (Metz *et al.*, 2016)), the convergence of GANs still remains a major problem. During the training progression, the feedback of D to G becomes meaningless, and if GANs continue to train past this point, the quality of the synthetic samples can be affected and ultimately collapse. Common variations of GANs cannot be trained as single-stream networks, and a necessary step is to define a training schedule for G and D separately, adding another layer of complexity. Although all deep learning models are sensitive to hyperparameter choices, Lucic *et al.* (2018) show all experimented GANs (including WGANs) are much more sensitive to these choices than VAEs. This can be a drawback in using GANs for scRNAseq generation since the hyperparameters may need to be re-tuned for every new dataset.

A.2. Variational Autoencoders

VAEs (Kingma and Welling, 2013, 2019) are generative models that jointly learn deep latent-variable and inference models. Specifically, VAEs are autoencoders that use variational inference to reconstruct the original data, having the ability to generate new data that is "similar" to those already in a dataset x. VAEs assume that observed data and latent representation are jointly distributed as $P_\theta(x, z) = P_\theta(x|z)P(z)$. In deep learning, the log-likelihood $P_\theta(x|z)$ is modeled through non-linear transformations, thus making the posterior probability distribution, $P_\theta(z|x) = \frac{P_\theta(x|z)P_\theta(z)}{P_\theta(x)}$, intractable. Due to the intractability of maximizing the expected log-likelihood of observed data over θ, $\mathbb{E}_{p(x)}\left[\log \int P_\theta(x, z)dz\right]$, the goal is to instead maximize the evidence lower bound (ELBO):

$$\underbrace{\mathbb{E}_{Q_{\gamma(x)}(z)}\left[\log\left(\frac{P_\theta(x|z)P(z)}{Q_{\gamma(x)}(z)}\right)\right]}_{\text{ELBO}(\theta,\gamma)} \leq \log P_\theta(x),$$

where $Q_{\gamma(x)}(z)$ is an auxiliary variational distribution (with parameters $\gamma(x)$) that tries to approximate the true posterior $P_\theta(z|x)$.

Note that when $Q_{\gamma(x)}(z) = P_\theta(z|x)$, the lower bound approaches the expected log-likelihood, therefore aiming to infer $P_\theta(z|x)$. We find the variational parameters γ for new inputs x using an inference network $Enc(\cdot)$ with parameters ϕ, such that $Enc_\phi(x) = \gamma(x)$. As shown by Zhao *et al.* (2017), this maximization problem can be written as

$$\arg\min_{\theta,\phi} -\text{ELBO}(\theta, \phi) = \arg\min_{\theta,\phi} \left(\mathbb{KL}\left(P(x)||P_\theta(x)\right) + \mathbb{E}_{P(x)}\left[\mathbb{KL}\left(Q_\phi(z|x)||P_\theta(z|x)\right)\right]\right),$$

where $Q_\phi(z|x)$ denotes the variational distributions $Enc_\phi(x)$ and $\mathbb{KL}(\cdot)$ denotes the Kullback–Leibler (KL) divergence. This yields to the objective function $\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{z\sim Q_\phi(z|x)}\left[\log P_\theta(x|z)\right] - \mathbb{KL}\left(Q_\theta(z|x)||P(z)\right)$. ELBO is the underlying objective function for this work, modified by Huang *et al.* (2018) to allow VAEs to train in an adversarial manner while maintaining key mathematical properties, as described in Section 3.

VAEs have also been criticized for generating samples that are "blurry" (adhering to an average of the data points), as opposed to sharp samples that GANs produce because of adversarial training. One possibility for this blurriness is the effect of maximum likelihood. That is, the model may assign high probability to other points rather than just the training data, and these points may include "blurry" samples/images. This issue has often been addressed by defining an adversarial training between the encoder and the decoder, as done in introspective VAEs (IntroVAEs) (Huang *et al.*, 2018), which we use in our framework. IntroVAEs are single-stream generative models that self-evaluate the quality of the generated images. They have been used mostly in computer vision, which have performed comparably to their GAN counterparts, in applications such as synthetic image generation (Huang *et al.*, 2018) and single-image super-resolution (Heydari and Mehmood, 2020).

A.3. Reconstruction Loss in VAEs

The expected negative reconstruction error L_{AE} is given by:

$$L_{AE} = \mathbb{E}_{z\sim Q(z|x)} \left[\log P_\theta(x|z)\right]$$

and we mention that we "choose" L_{AE} to be the mean squared error (MSE) between the training cells and reconstructed cells (as it is a typical choice). However, it is important to formally derive the reconstruction loss, since the true reconstruction loss is not MSE alone, as shown below. For simplicity, we normalize the expected reconstruction error by the number of samples n as

$$L_N = \frac{1}{n}\mathbb{E}_{z\sim Q(z|x)} \left[-\log P_\theta(x|z)\right],$$

and let $d(x, y) = \|x - y\|_2^2$. Moreover, for simplicity, we assume that $\sigma_\theta^2(z) = \sigma_\theta^2\mathbb{I}$. Now given the Gaussian assumption on the likelihood $P_\theta(x|z)$, we have the normalized reconstruction loss as:

$$L_N = \frac{1}{n}\mathbb{E}_{z\sim Q(z|x)} \left[\frac{1}{2}\left(\log(2\pi\sigma_\theta^2) + \frac{d\left(x, \mu_\theta(z)\right)}{\sigma_\theta^2}\right)\right].$$

Defining $MSE = \frac{1}{n}\mathbb{E}_{z\sim Q(z|x)} d(x, \mu_\theta(z))$ and substituting in the above expression yields:

$$L_N = \frac{1}{2}\left(\log(2\pi\sigma_\theta^2) + \frac{MSE}{\sigma_\theta^2}\right) = \alpha + \beta MSE, \quad \text{for } \alpha, \beta \in \mathbf{R}.$$

Now we can explicitly assuming that $\sigma_\theta^2 = \frac{1}{2}$ will give us:

$$L_N = \frac{\log(\pi)}{2} + MSE.$$

We can see that choosing MSE as the reconstruction loss is a simplification of the true reconstruction loss, which could potentially be sub-optimal in some cases. However, it is common to take MSE as the reconstruction loss when training a VAE model with a Gaussian Likelihood function [e.g. as done in Huang *et al.* (2018) and Daniel and Tamar (2021)].

Supplementary Material B : Data Post-Processing

B.1. Post-Processing

After generating a count matrix with a generative model (e.g. ACTIVA or scGAN), we add the gene names (from the real data) and save as a Scanpy/Seurat object. We then use Seurat to identify 3000 highly variable genes through the use of variance-stabilization transformation (VST) [Hafemeister and Satija (2019)], which applies a negative binomial regression to identify outlier genes. The shared highly variable genes are then used for integration [Stuart *et al.* (2019)], which allows for biological feature overlap between different datasets in order to perform the downstream analyses presented in this work. Next, we perform a gene-level scaling, i.e. centering the mean of each feature to zero and scaling by the standard deviation. The feature space is then reduced to 50 principal components, followed by Uniform Manifold Approximation and Projection (UMAP) [McInnes *et al.* (2018)] and t-distributed Stochastic Neighbor Embedding (t-SNE) [van der Maaten and Hinton (2008)]. As noted by Marouf *et al.* (2020), analysis with lower-dimensional representations have two main advantages: (i) most biologically relevant information is captured while noise is reduced and (ii) statistically, it is more acceptable to use lower dimensional embeddings in classification tasks when samples and features are of the same order of magnitude, which is often the case with scRNAseq datasets (such as the ones we used). Lastly, we use Scater [McCarthy *et al.* (2017)] to visualize the datasets.

Supplementary Material C : Complete Computational Environment

Development and testing were done on Accelerated Computing EC2 instances (`p3.2xlarge` and `p3.8xlarge`) of Amazon Web Services. All requirements and dependencies are automatically installed by our package and are listed in a requirements file, but for the sake of completeness, they are as follows: Python v3.7.6, PyTorch v1.5.1, NumPy v1.18.5, SciPy v1.4.1, Pandas v1.2.0, Scanpy v1.6.0, AnnData v0.7.5, and Scikit-learn v0.24.0. For data pre- and post-processing, we used LoomPy v3.0.6, SeuratDisk v0.0.0.9013, Seurat v3.2.3, scater v1.16.2, , and R v4.0.3. For evaluation of dropout rates we used M3Drop v1.18.0, ggplot2 v3.3.2. Differential state analysis was performed with muscat v1.6.0. The scGAN package was run in a Docker container (using the provided dockerfile at `https://github.com/imsb-uke/scGAN/tree/master/dockerfile`). Reported training times for ACTIVA/scGAN were averages of 5 times on a single NVIDIA-Tesla V100 GPU (Table S1). Inference times were averages of 5 measurements on (i) V100 GPU (GPU time) and (ii) 2.3 GHz Quad-Core Intel Core i7 (on a 2020 MacBook Pro).

Supplementary Material D : GPU Training Times

Table S1. Training time (in seconds) on 68K PBMC for ACTIVA and scGAN on 1 NVIDIA Tesla V100 GPU. We trained each model 5 times under the same conditions to find an average training time. We have not included cscGAN times since training that model took longer than scGAN. We can see that ACTIVA trains much faster (6.3 times faster on average). scGAN and cscGAN were run in a Docker container (by Marouf et al.) using the Dockerfile located at `https://github.com/imsb-uke/scGAN/tree/master/dockerfile`).

Iteration	ACTIVA	scGAN	cscGAN
1	25968.2447	164225.5618	175978.2588
2	26218.7306	165308.7142	176041.7922
3	25935.2738	164391.8113	176318.0701
4	26091.8337	165281.5137	175941.7097
5	25915.6733	164988.1371	175792.6410
Average	26025.95 (≈7.2 hours)	164839.14 (≈45.7 hours)	176014.49 (≈48.8 hours)

Table S2. Training time (in seconds) on Brain Small for ACTIVA and scGAN on 1 NVIDIA Tesla V100 GPU. We trained each model 5 times under the same conditions to find an average training time. ACTIVA trains approximately 17 times faster than scGAN. scGAN and cscGAN were run in a Docker container (using the Dockerfile located at `https://github.com/imsb-uke/scGAN/tree/master/dockerfile`).

Iteration	ACTIVA	scGAN	cscGAN
1	8277.4867	142371.9793	145980.3154
2	7922.1005	141103.8196	145952.2580
3	8107.3591	143008.7401	145261.1851
4	7983.4031	142532.3804	146076.8253
5	8084.2473	142173.5900	146009.3626
Average	8074.91 (≈2.2 hours)	142238.10 (≈39.5 hours)	145855.98 (≈40.5 hours)

Table S3. Training time (in seconds) on NeuroCOVID for ACTIVA and scGAN on 1 NVIDIA Tesla V100 GPU. We trained each model 5 times under the same conditions to find an average training time. ACTIVA trains approximately 6 times faster than scGAN. scGAN and cscGAN were run in a Docker container (using the Dockerfile located at `https://github.com/imsb-uke/scGAN/tree/master/dockerfile`).

Iteration	ACTIVA	scGAN	cscGAN
1	29604.6613	184421.5509	187618.5119
2	29351.4168	183863.5722	187440.2821
3	29719.3984	184249.4072	188414.8095
4	29492.1603	183120.8326	187922.5381
5	29575.2929	183814.1538	187924.3118
Average	29548.58 (≈ 8.2 hours)	183749.10 (≈ 51.0 hours)	187864.09 (≈ 52.1 hours)

Supplementary Material E : Runtime Analysis

As mentioned in the main manuscript, ACTIVA trains much faster than the GAN-based models due to its architecture. ACTIVA's runtime depends on a number of factors, such as the complexity of the dataset, number of genes, sparsity and the hardware used for training −even using different GPUs will result in training time differences. However, since ACTIVA uses a batch training (as done in almost all deep learning models), out of memory (OOM) errors are more controlled, and the runtime scales linearly (at worst). Here, we aim to demonstrate this fact with so-called "corner" cases. Due to the nature of scRNAseq experiments, we believe that it is unlikely that ACTIVA will be trained with 10^5 cells or more, but we provide the runtimes for 10^5, 2×10^5 and 5×10^5 for reference. Due to the computational costs, we did not replicate the runtime experiment for scGAN and cscGAN. However, the runtimes can be extrapolated from our previous results and Marouf *et al.* (2020).

To construct the mentioned corner cases, we generated random data $X_{runtime} \in \mathbb{R}^{n \times d}$, where d is the number of genes in 68K PBMC, and n is the number of cells ($n \in \{100000, 200000, 500000\}$). Next, we measured the percentage of non-zero entries in the 68K PBMC data (often referred to as density). We found that the 68K PBMC had a density of about 3%. However, we set the density of $X_{runtime}$ to be 30% to create a "worst-case" for training. Since we were only interested in runtime of ACTIVA for $X_{runtime}$ and not the data generation quality for this random data, we used all of $X_{runtime}$ for training. The training data dimensions were 100000×17789, 200000×17789 and 500000×17789. We repeated training on each dataset five times, and present the runtimes in Table S4.

Table S4. Runtime analysis for ACTIVA with various number of cells. To simulate the "worst-cases", we chose the same number of genes as in 68K PBMC (17789), and we took the density of the count matrix to be 30% (compared to about 3% for 68K PBMC). We then trained ACTIVA for various number of cells for 5 iterations, which we present in this table. The reported times are measured in seconds.

Iteration	100K	200K	500K
1	33143.3375	61270.0890	129722.8039
2	33053.8834	61255.4751	126041.6814
3	32931.2144	61183.5369	124377.9299
4	32995.6469	61230.1366	130018.3193
5	33181.0439	61091.5801	128588.0704
Average	33061.0252 (≈ 9.1 hours)	61206.1635 (≈ 17.0 hours)	127749.7609 (≈35.48 hours)

Supplementary Material F : Results on NeuroCOVID Data

In this section, we present our qualitative and quantitative results on our third dataset: NeuroCOVID.

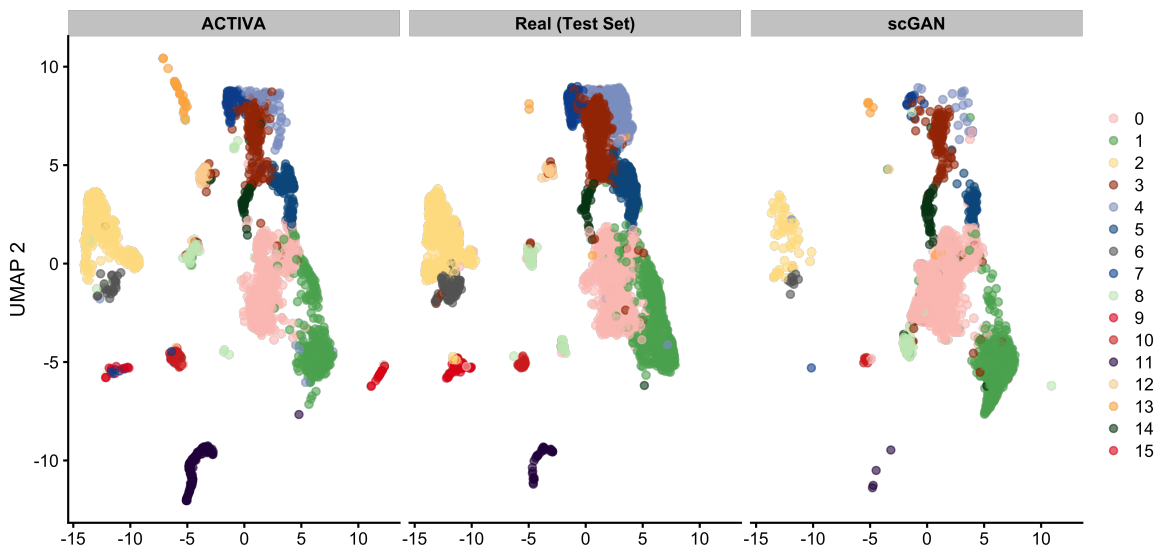

Fig. S1: UMAP plot of ACTIVA generated cells compared with test set and scGAN generated cells, colored by clusters for NeuroCOVID. ACTIVA's cell-type conditioning encourages the model to generate more cells per cluster, meaning that ACTIVA will generate more cells from the rare populations as shown in the above visualization.

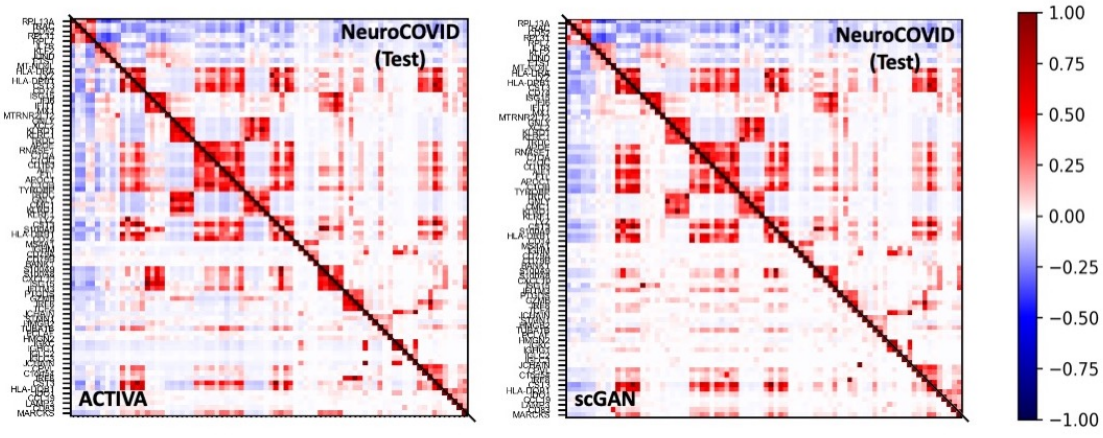

Fig. S2: We performed the correlation analysis described in Section 5.3, where we looked at the correlation of top genes 5 genes from each cluster, and measured the pair-wise correlation of those genes in each dataset (test set, ACTIVA and scGAN). As presented in this figure, ACTIVA resembles closer relationship to the real data than scGAN, achieving correlation discrepancy (CD) score of 4.19 in comparison to scGAN's 7.31 (lower CD is better).

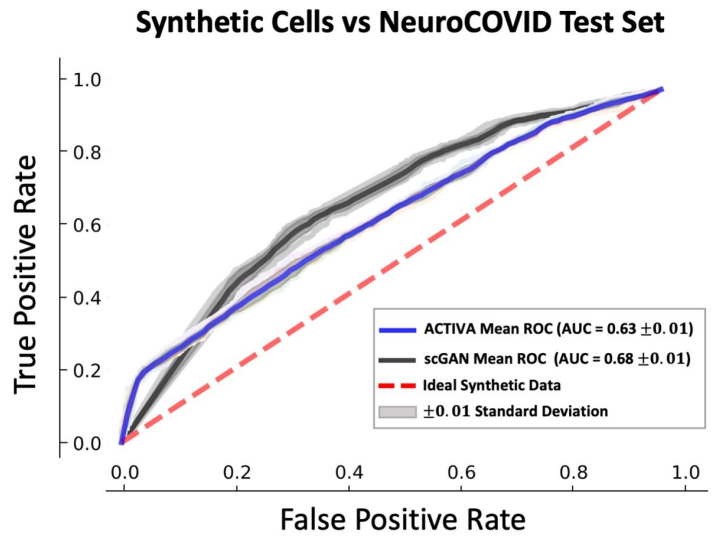

Fig. S3: Random Forest identification of synthetic data (generated from ACTIVA and scGAN) against the real data (test set). As described in the main manuscript, AUC's closer to 0.5 indicate better performance for the generative model, since that means that the classifier could not properly distinguish the difference between real cells and generated cells.

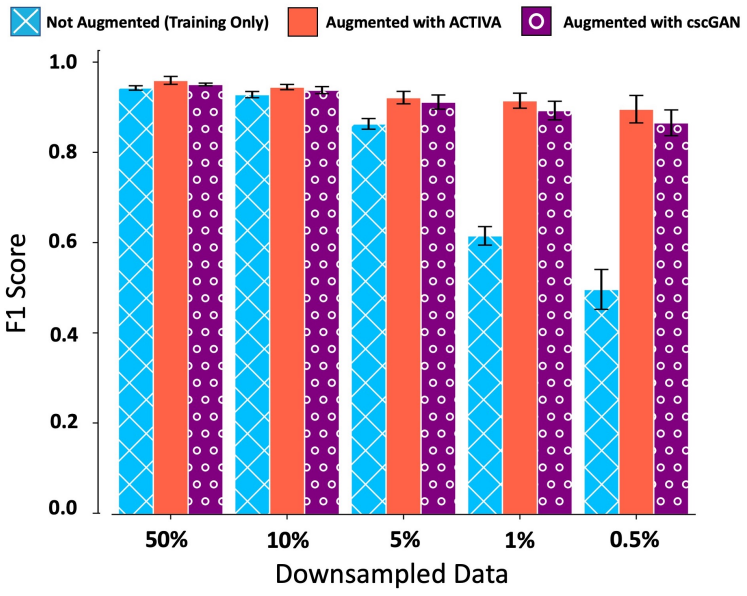

Fig. S4: Mean F1 scores of Random Forest (RF) classifier for NeuroCOVID training data (with no augmentation), shown in blue, and training data augmented with ACTIVA (shown in red) and scGAN (shown in purple), respectively. Error bars indicate the range for five different random seeds for sub-sampling cluster 7 cells in NeuroCOVID data.

Supplementary Material G : Choosing Adversarial Constant m

Ensuring the numerical balance between the KL divergence regularization of real and fake samples is crucial to ACTIVA's sample quality. Therefore an m that is extremely large or small could result affect the quality of the generated cells. The adversarial training in IntroVAE is similar to Energy-based GANs [Zhao *et al.* (2016)]. The following are two strategies we followed for choosing an appropriate value of m, based on Zhao *et al.* (2016):

1. An effective strategy is to train the model as VAE for n epochs and track the minimized $\mathbb{KL}$ divergence. This will provide an estimate of the capacity of Gen's reconstruction of single cells without a critic's input. Then, set m to be a value close to minimized $\mathbb{KL}$ divergence after n epochs. This feature is already implemented in our package, with a default VAE-only training of 10 epochs. In practice, we found that values of m roughly close to this minimized divergence performed well.
2. Another strategy for choosing m can be a rough grid-search starting from large values of m (which could be the upper bound of $\mathbb{KL}$ values) and gradually going to 0.

Supplementary Material H : Network Architecture

In this section, we present the architecture of our model with the input $x \in \mathbb{R}^M$. Latent vectors of our model live in a 128-dimensional space, but for the sake of generality, we assume $z \in \mathbb{R}^D$. In the encoder and generator networks, Adam (Kingma and Ba, 2015) optimizer is used with a learning rate $lr = 0.0002$, and moving averages decay rates $\beta_1 = 0.9$, $\beta_2 = 0.999$. Gradients are calculated on mini-batches of size 128 with the adversarial constant $m = 110$. We describe each component in more detail in H.1..

H.1. Training and Inference Procedure

Unlike GANs, our model does not require a training schedule. The cell-type classifier in ACTIVA can be pre-trained or trained simultaneously with Enc and Gen. To start the adversarial training and pick an appropriate m, we first train the model as a VAE for 10 epochs while training ACTINN in parallel (these options are readily available and adjustable in our package). After the initial warm-up, we train the IntroVAE component for 600 epochs with $\alpha_1 = 1$ and $\alpha_2 = 0.5$ (from Eq. (4)-(5)). We also provide an option for dynamic weight-balancing using SoftAdapt that would be useful in the case of a posterior collapse (which did not occur in our experiments).

For inference, we input a random noise tensor sampled from a multi-variate Gaussian to the trained generator. For cell-specific generation, outputs are automatically filtered through the trained classifier to produce the desired sub-populations on demand. Tutorials and notebooks on training and inference are available via link provided in abstract.

H.2. Encoder Network

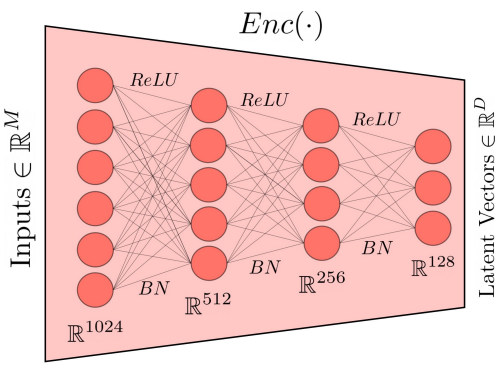

Fig. S5: The encoder network of ACTIVA.

Enc consists of fully connected layers, with Rectified Linear Units ($ReLU$) [Nair and Hinton (2010)] as the activation between two layer, where we also perform batch normalization operation in [Ioffe and Szegedy (2015)] (denoted as BN) after $ReLU$. The input to the network is $x \in \mathbb{R}^M$, which goes through the network with layers $\{1024, 512, 256, 128\}$, as shown in Fig. S5. Adam [Kingma and Ba (2015)] optimizer is used with a learning rate $lr = 0.0002$, and moving average decay rates $\beta_1 = 0.9$, $\beta_2 = 0.999$. Gradients are calculated on mini-batches of size 128, with the adversarial constant $m = 110$.

H.3. Generator Network

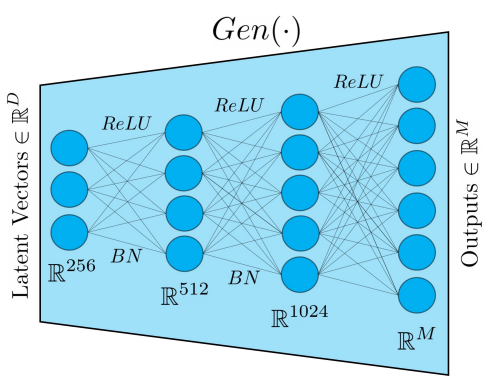

Fig. S6: Architecture of ACTIVA's generator network.

The generator network mirrors the encoder network, consisting of an input latent vector $z \in \mathbb{R}^D$ going through the layers $\{256, 512, 1024, M\}$ as shown in Fig. S6. Note that in the last layer of the generator (mapping from 1024 to M), we use $ReLU$ without BN. This is because we want to ensure that all generated values are non-negative. Similar to the encoder, we optimize the network using Adam with a learning rate $lr = 0.0002$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. Gradients are calculated on 128-cell mini-batches.

H.4. Automated Cell Type Network (ACTINN)

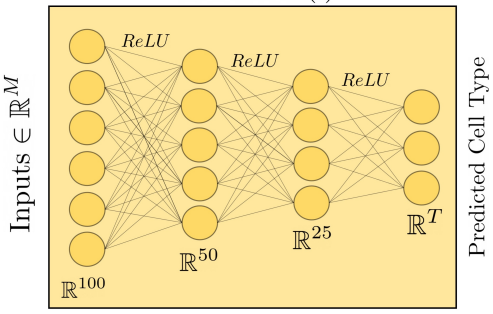

Fig. S7: Automatic cell-type identification network of ACTIVA, which is ACTINN.

Ma and Pellegrini (2019) use a fully connected neural network architecture for supervised classification of cell-types. scRNAseq data is usually high dimensional and often sparse, making neural network a promising method for analyzing such data. We implemented ACTINN in PyTorch, and used it for identifying cell-types and conditioning ACTIVA. Our implementation has same architecture as in Ma and Pellegrini (2019) which was implemented in TensorFlow (S7). Cross entropy is used to measure the loss between the predicted classes and the actual cell types. Optimization is done with Adam and an exponential "staircase" decay is used, with initial learning rate being $lr = 0.0001$ and a decay rate of 0.95 applied after every 1000 optimization steps. Our implementation of ACTINN in PyTorch differs from the original implementation in two ways: (1) we do not use a SoftMax layer between the last hidden layer and the output layer (due to the implementation of cross-entropy in PyTorch), and (2) we train for fewer number of epochs (between 5-10 as opposed to 50 epochs in the original implementation). Although we train for fewer epochs that Ma and Pellegrini (2019), we did not notice a drop in the accuracy of our model; that is, our results on 68K PBMC closely match the results found by Abdelaal *et al.* (2019) (results are shown in Tables S6-S7). Gradients are calculated on mini-batches of size 128.

Supplementary Material I : Downsampling and Data Augmentation

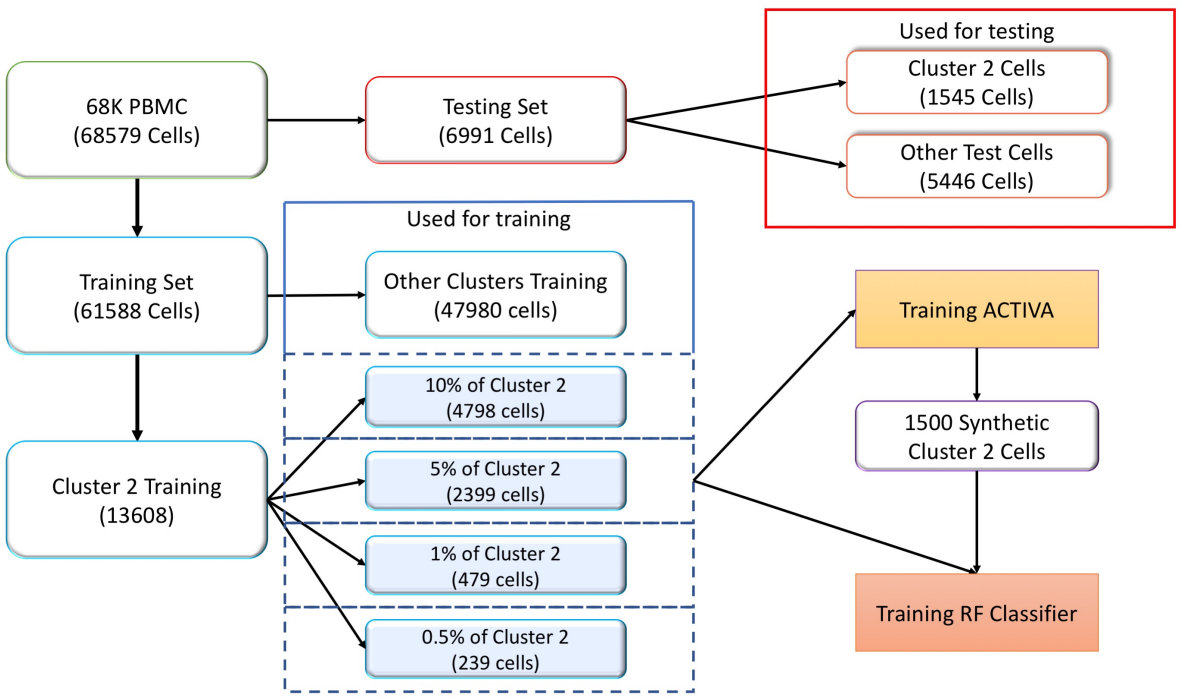

Fig. S8: Downsampling process for evaluating the impact of data augmentation with ACTIVA (Test splits shown in red frames and Training splits are in blue frames, with ACTIVA generated in purple frame). We first separate the test set from the training set, and subsequently label cluster 2 cells to differentiate them from all other clusters. Cells from all other clusters (besides cluster 2) are used in all training and testing modes. For cluster 2 cells, we randomly subsample a fraction of the cells (10%, 5%, 1% or 0.5%) and use this subset in addition to all other cells to train ACTIVA. In other words, ACTIVA and the RF will include (i) training data from all other clusters and (ii) one of the downsampled version of cluster 2 cells (highlighted in light blue). For the performance evaluation of RF without data augmentation ("no-augmentation"), we only use the desired cluster 2 subset and all other training cells to train the classifier. For training mode ACTIVA augmentation, we generate 1500 cells for data augmentation, and add to the training cells we used in "no-augmentation" mode. So for training the RF in augmentation mode, we use (i) training data from all other clusters, (ii) one of the downsampled version of cluster 2 cells (highlighted in light blue) and (iii) 1500 ACTIVA generated cells [trained on the same donwsampled data in (ii)].

Supplementary Material J : Additional Results

J.1. Dropout Rate

A common feature in scRNAseq data is technical zero counts (referred to as “dropouts") which can arise from low RNA capture. To analyze the dropout rates as a function of mean gene expression for our generated data, we use functions from M3Drop (Andrews and Hemberg, 2018) to extract the observed dropout rates for ACTIVA and scGAN and compare those with the raw data for 68K PBMC, Brain Small and NeuroCOVID. Our analysis showed that our model produced a lower dropout rate for higher mean expressions, which can we attribute to the assumption on the prior distribution. Our dropouts were very close to scGAN’s for 68K PBMC, while for NeuroCOVID, our data resembled the real data more closely. In the Brain Small dataset, we observed a steeper dropout rate for ACTIVA in relation to the real dataset. We hypothesize that the steeper dropout rate is due to fewer samples in the training data, which was about 4 times fewer than 68K PBMC and NeuroCOVID.

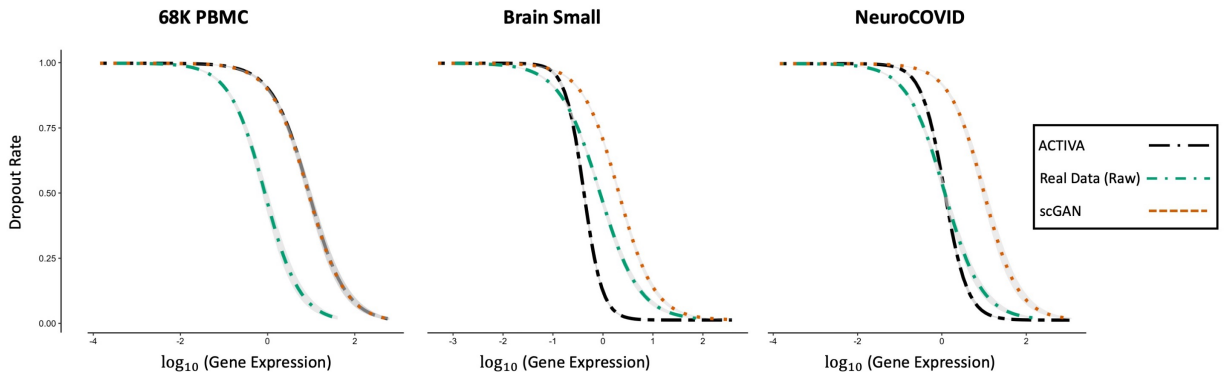

Fig. S9: Observed dropout rates as a function of $\log_{10}$(Gene Expression) for all three datasets. The dropout rate for ACTIVA is shown in Black (two-dashed line), Real in green (dot-dashed line), scGAN in Orange(dotted line) with the 95% confidence interval for each line shown in grey. Data was fit using a binomial generalized linear model.

J.2. Differential States in Clusters

As another evaluation metric for ACTIVA’s generative quality, we perform a differential state (DS) analysis using muscat (Crowell *et al.*, 2020). DS accounts for for sample-to-sample as well as cell-to-cell variability, allowing us to draw conclusions extrapolate to the samples rather than cells (Crowell *et al.*, 2019). In our case, we measure the sample-to-sample variability to be between generated cells and the real cells (test data). The idea is that if the generative models generate realistic samples, then there should be *fewer DS genes* in relation to the real data. To perform the differential state (DS) analysis, we sample each data (ACTIVA, real test set and scGAN) without replacement to create 5 pseudo-replicates for each dataset, which formed the “pseudobulk data". From this pseudobulk, ACTIVA was directly compared with the real (referred to as ACTIVA-Real in Fig. S10) and scGAN was compared with the real (scGAN-REAL). For all three datasets, we find that the generated-real cluster pairs contains only a few DS genes in comparison to the real data, indicating a high quality of synthetic data. However, ACTIVA has fewer DS genes for NeruoCOVID and Brain Small compared to scGAN, while scGAN having fewer DS genes in the 68K PBMC clusters.

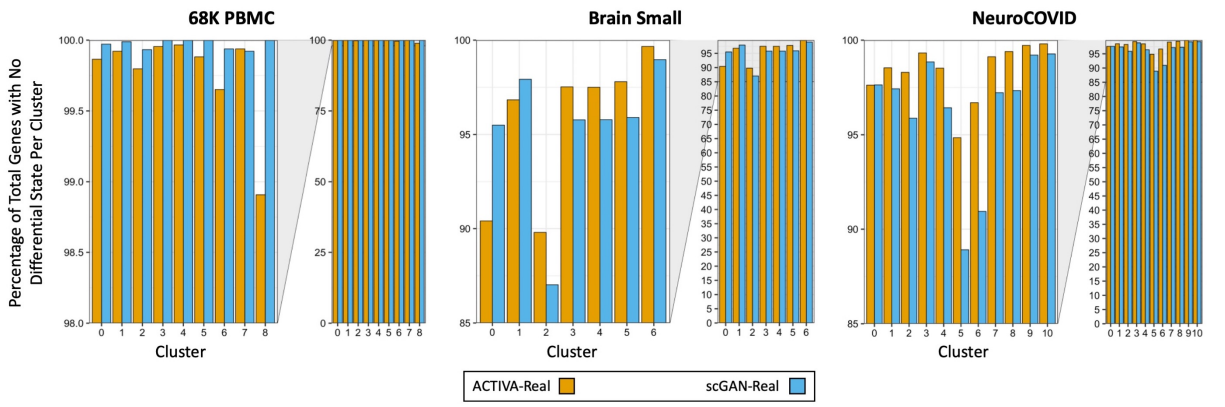

Fig. S10: Barplots displaying the percent of genes with no differential state per cluster for all three datasets. Values closer to 100% indicate a better match with the real data. In orange, ACTIVA is compared with the real data (test set), and in blue, scGAN is compared with the real data (test set). Our results show a high null differential state per clusters for all three datasets. The statistical analyses were performed in cluster with sufficient number of cells (clusters with too few cells are not shown here).

J.3. Accuracy of the Classifier on Each Dataset

As mentioned in the main manuscript, the sub-population cell generation depends on correctly classifying the cell-types. Here, we show that ACTIVA's classifier, ACTINN, can classify rare-cell population, with training cells as few as 71 cells (see Table S7). We did observe that in some cases, if the number of training samples for a specific cluster is extremely low, then ACTIVA does not learn that cell-type (as it is expected from any machine learning model); for example, the classifier does not learn the 10th cell-types of the PBMC data, since there were only 19 training cells available (out of 61K training cells). However, for cluster 2 cell population, having 50 cells resulted in an F1 score of 0.74. This was useful for studying the impact of data augmentation with ACTIVA, as described in Section 5 of the main manuscript.

Table S5. Accuracy of ACTIVA's classifier network (ACTINN) on the test sets of Brain Small, 68K PBMC and NeuroCOVID. Three metrics were used: (1) Accuracy-number of correct predictions over all predictions; (2) F1 Score (Non-Weighted): unweighted mean of per-label accuracy (not counting for cell-type imbalance); and (3) Weighted F1 Score: per-type accuracy but weighted by the number of cells for each cell-type.

Test Set	Accuracy	F1 Score (Non-Weighted)	Weighted F1 Score
Brain Small	0.9649	0.9674	0.9654
68K PBMC	0.9223	0.7448	0.9216
NeuroCOVID	0.9790	0.9697	0.9790

Table S6. Here we present the accuracy of ACTIVA's classifier network on the test and training set for Brain Small dataset (see main text Section 4.3 and Fig. 3).

Cluster	Testing Cells	Test Precision	Test Recall	Test F1-Score	Training Cells	Train Precision	Train Recall	Train F1-Score
0	978	0.97	0.97	0.97	8808	0.99	1.00	0.99
1	304	0.98	0.99	0.98	2738	0.99	1.00	0.99
2	271	0.90	0.93	0.92	2439	0.98	0.99	0.98
3	182	0.99	0.96	0.97	1646	1.00	0.98	0.99
4	141	0.99	0.94	0.97	1271	1.00	0.99	0.99
5	59	0.98	0.98	0.98	535	1.00	0.99	1.00
6	35	1.00	0.94	0.97	323	1.00	0.97	0.98
7	27	1.00	0.93	0.96	243	0.99	0.99	0.99

Table S7. Here we present the accuracy of ACTIVA's classifier network on the test and training set for 68K PBMC dataset(see main text Section 4.3 and Fig. 3).

Cluster	Testing Cells	Test Precision	Test Recall	Test F1-Score	Training Cells	Train Precision	Train Recall	Train F1-Score
0	1791	0.89	0.90	0.90	15768	0.99	1.00	1.00
1	1545	0.88	0.87	0.88	13608	1.00	0.99	1.00
2	1515	0.93	0.96	0.95	13344	1.00	1.00	1.00
3	697	0.91	0.87	0.89	6145	1.00	0.99	1.00
4	483	0.99	0.98	0.99	4258	1.00	0.99	1.00
5	466	0.97	0.97	0.97	4105	1.00	1.00	1.00
6	413	1.00	1.00	1.00	3644	1.00	1.00	1.00
7	71	0.98	0.82	0.89	626	1.00	0.95	0.97
8	8	0.00	0.00	0.00	71	1.00	0.68	0.81
9	2	0.00	0.00	0.00	19	0.00	0.00	0.00

J.4. Gene Expressions

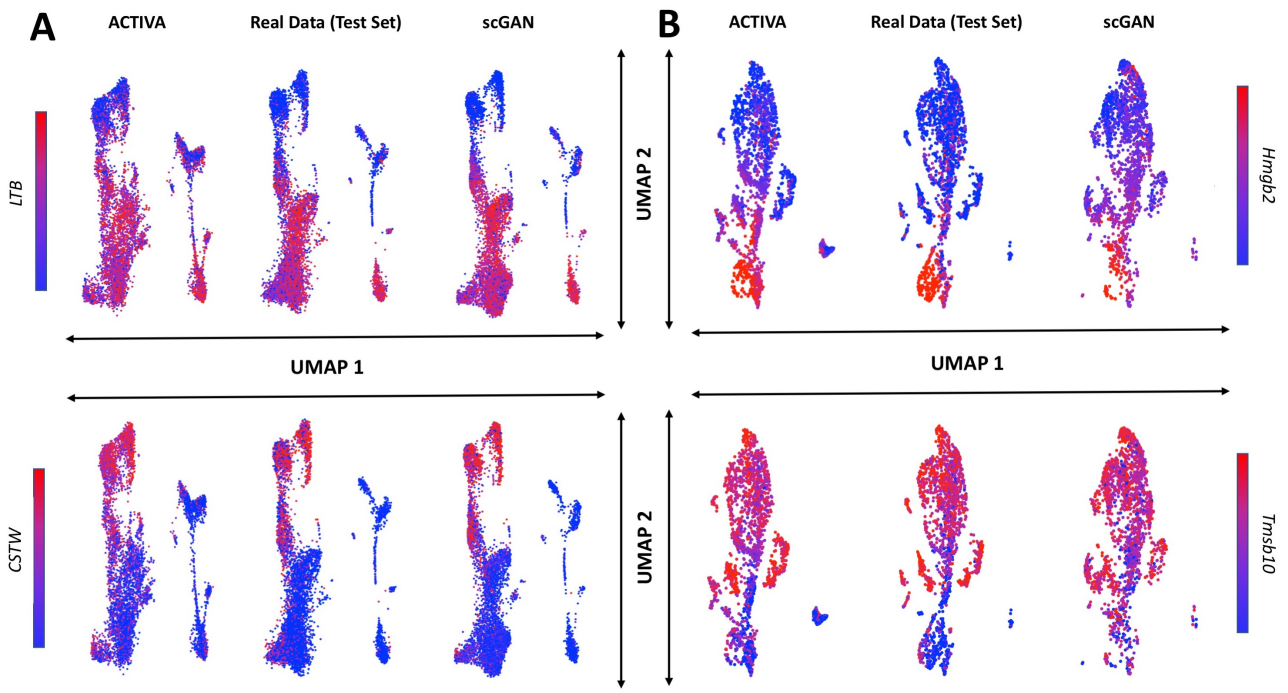

Fig. S11: UMAP of Synthetic Cells (generated by ACTIVA and scGAN with a subset of real data as training data) compared to real data (not used in training) colored by gene expression. Column (A) . **Column A**: here we present results from 68K PBMC data with a UMAP of 6991 cells generated by ACTIVA and scGAN, respectively, along with the test set, colored by the gene expression of two markers genes. **Column B**: we show the results on Brain Small test set and 1997 ACTIVA and scGAN generated cells, colored by the gene expression of two marker genes. For both datasets, we see that cells generated by ACTIVA resemble the real data well while more diversity is present among the generated cells.

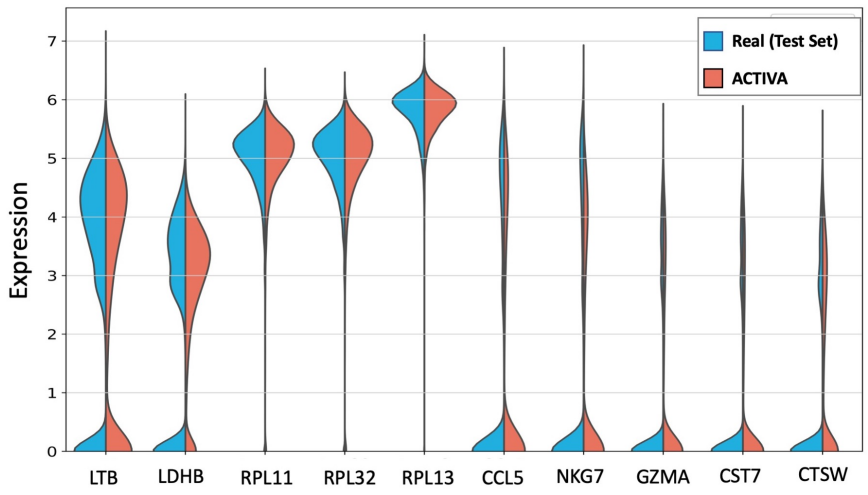

Fig. S12: **Logarithmic expression of the top marker genes**. First 5 are for cluster 1 and last 5 for cluster 2 in 68K PBMC test set (in blue) and ACTIVA generated cells (in red). The similarity between the distributions indicates that ACTIVA has learned the underlying marker gene expression.

J.5. Manifold Analysis

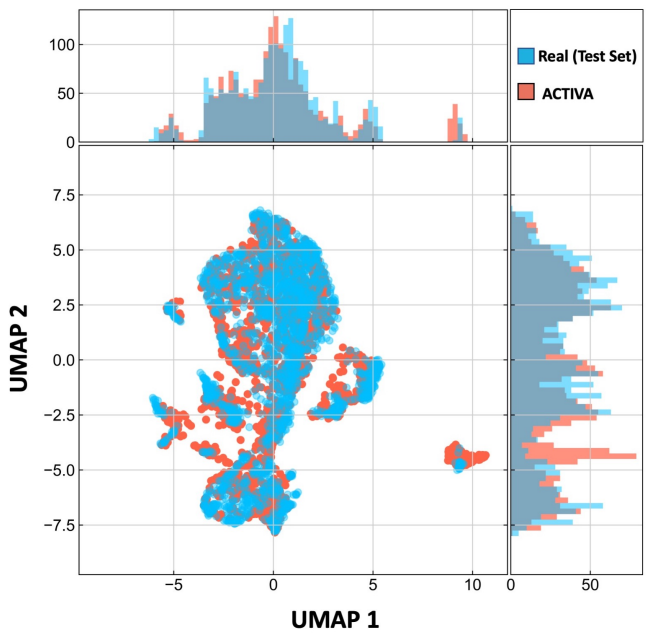

Fig. S13: UMAP of cells generated by ACTIVA compared to real test cells (not used in training) from 20K Brain Small. The histograms on top and right of the UMAP plot display the counts of cells on the horizontal and vertical axis, respectively. This figure shows that ACTIVA has learned the underlying distribution of the real data while having some diversity in the generated samples (which is desired for generative models).

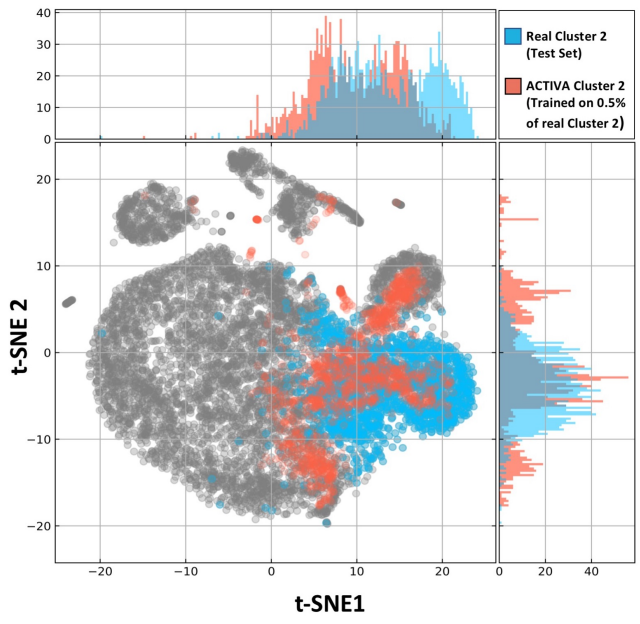

Fig. S14: t-SNE of all the cells in the test set (in grey), real cluster 2 cells (in blue), and ACTIVA generated cells when trained with only 0.5% of cluster 2 cells (in red). This figure illustrates that ACTIVA learns to generate a sub-population even when it is trained on 239 cells (out of 7915 training cells). This qualitative evaluation, combined with the results shown in the manuscript, show promising application of ACTIVA for improving downstream classification of rare populations.

References

Abdelaal, T. *et al.* (2019). A comparison of automatic cell identification methods for single-cell RNA sequencing data. *Genome Biology*, **20**(1), 194.

Andrews, T. S. and Hemberg, M. (2018). M3Drop: dropout-based feature selection for scRNASeq. *Bioinformatics*, **35**(16), 2865–2867.

Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *arXiv:1701.04862*.

Arjovsky, M. *et al.* (2017). Wasserstein generative adversarial networks. volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia. PMLR.

Crowell, H. L., Soneson, C., Germain, P.-L., Calini, D., Collin, L., Raposo, C., Malhotra, D., and Robinson, M. D. (2019). On the discovery of population-specific state transitions from multi-sample multi-condition single-cell rna sequencing data. *bioRxiv*.

Crowell, H. L., Soneson, C., Germain, P.-L., Calini, D., Collin, L., Raposo, C., Malhotra, D., and Robinson, M. D. (2020). muscat detects subpopulation-specific state transitions from multi-sample multi-condition single-cell transcriptomics data. *Nature Communications*, **11**(1), 6077.

Daniel, T. and Tamar, A. (2021). Soft-introvae: Analyzing and improving the introspective variational autoencoder.

Dziugaite, G. K. *et al.* (2015). Training generative neural networks via maximum mean discrepancy optimization. *arXiv:1505.03906*.

Engel, J. *et al.* (2019). Gansynth: Adversarial neural audio synthesis. *arXiv:1902.08710*.

Fedus, W. *et al.* (2018). MaskGAN: better text generation via filling in the_. *arXiv:1801.07736*.

Goodfellow, I. *et al.* (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Hafemeister, C. and Satija, R. (2019). Normalization and variance stabilization of single-cell RNA-seq data using regularized negative binomial regression. *Genome Biology*, **20**(1), 296.

Heydari, A. A. and Mehmood, A. (2020). SRVAE: super resolution using variational autoencoders. In *Pattern Recognition and Tracking XXXI*, volume 11400, page 114000U. International Society for Optics and Photonics.

Huang, H. *et al.* (2018). IntroVAE: Introspective variational autoencoders for photographic image synthesis. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 52–63. Curran Associates, Inc.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *CoRR*, **abs/1312.6114**.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, **12**(4), 307–392.

Larsen, A. B. L. *et al.* (2016). Autoencoding beyond pixels using a learned similarity metric. volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566, New York, New York, USA. PMLR.

Liu, Q. *et al.* (2019). hicGAN infers super resolution Hi-C data with generative adversarial networks. *Bioinformatics*, **35**(14), i99–i107.

Lucic, M. *et al.* (2018). Are GANs created equal? A large-scale study. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 698–707, Red Hook, NY, USA. Curran Associates Inc.

Ma, F. and Pellegrini, M. (2019). ACTINN: automated identification of cell types in single cell RNA sequencing. *Bioinformatics*, **36**(2), 533–538.

Marouf, M. *et al.* (2020). Realistic in silico generation and augmentation of single-cell RNA-seq data using generative adversarial networks. *Nature Communications*, **11**(1), 166.

McCarthy, D. J. *et al.* (2017). Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics*, **33**(8), 1179–1186.

McInnes, L. *et al.* (2018). UMAP: Uniform manifold approximation and projection for dimension reduction.

Metz, L. *et al.* (2016). Unrolled generative adversarial networks. *CoRR*, **abs/1611.02163**.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

Stuart, T. *et al.* (2019). Comprehensive integration of single-cell data. *Cell*, **177**(7), 1888 – 1902.e21.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, **9**(86), 2579–2605.

Wang, Z. *et al.* (2019). Generative adversarial networks in computer vision: A survey and taxonomy. *arXiv:1906.01529*.

Zhao, J. *et al.* (2016). Energy-based generative adversarial network. *arXiv:1609.03126*.

Zhao, S. *et al.* (2017). Towards deeper understanding of variational autoencoding models. *arXiv:1702.08658*.

Zhu, J.-Y. *et al.* (2016). Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pages 597–613. Springer.