Dependency-aware deep generative models for multitasking analysis of spatial omics data. Report

Anastasiia Kazovskaia (SID: 20992340)

December, 12

Introduction

In the paper Tian et al. (2024), the authors present **spaVAE**, a dependency-aware deep generative spatial variational autoencoder (VAE) designed to probabilistically model count data while accounting for spatial correlations. Unlike a classical VAE, which usually assumes only a Gaussian prior, spaVAE incorporates a hybrid embedding that combines a Gaussian process prior with a Gaussian prior to explicitly capture spatial relationships among spots. By optimizing the parameters of deep neural networks, spaVAE approximates the underlying distributions of spatially resolved transcriptomics (SRT) data. These approximated distributions enable spaVAE to support a range of key analytical tasks in SRT data analysis, including dimensionality reduction, visualization, clustering, denoising, differential expression analysis, spatial interpolation, resolution enhancement, and the identification of spatially variable genes.

Essentially, spaVAE is a hierarchical Bayesian model, where conditional distributions are parameterized by deep neural networks. The model offers several key advantages:

- **Adaptive spatial dependency modeling:** Spatial dependencies are incorporated directly into the learning process, with their strength adaptively learned from the data
- **Hybrid embedding approach:** By combining Gaussian process and standard Gaussian embeddings, spaVAE captures both spatially dependent and independent variations in SRT data. This enables efficient characterization of the underlying data distribution
- **Direct modeling of gene count data:** spaVAE employs a negative binomial reconstruction loss, which effectively addresses over-dispersion and library size variations
- **Computational efficiency:** By leveraging sparse Gaussian process regression as in Hensman et al. (2013), spaVAE accelerates computation, enabling scalability to large datasets

With these features, spaVAE provides a powerful and efficient framework for comprehensive SRT data analysis.

Methods

Dependency-aware variational autoencoder

Denote the number of dimensions of latent representations as D, the first L dimensions follow a Gaussian process prior, and the other $D-L$ dimensions follow a standard Gaussian prior. Next, denote the sample size as N, the spatial locations as $\boldsymbol{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}]^T \in \mathbb{R}^{N\times 2}$, and the gene count matrix as $\boldsymbol{y} = [\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_N}]^T \in \mathbb{R}^{N\times G}$, where G is the number of the expressing genes. The objective is to learn a model that could infer low-dimensional latent representation $\boldsymbol{z}$ and generate $\boldsymbol{y}$ conditioned on auxiliary data $\boldsymbol{x}$. In the framework of VAE, the evidence lower bound (ELBO) can be written as:

$$ELBO = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}[\log p(\boldsymbol{y}|\boldsymbol{z})] - \beta KL\left(q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})\|p(\boldsymbol{z})\right), \tag{1}$$

where $\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}[\log p(\boldsymbol{y}|\boldsymbol{z})]$ can be considered as the reconstruction loss, and a hyperparameter β is used to control the weight of Kullback–Leibler (KL) divergence loss.

Gene likelihood

To characterize the gene count data, in particular, to model mRNA count data, a negative binomial (NB) reconstruction loss is used. The variational inference models the count matrix $\boldsymbol{y}$ likelihood by the NB distribution:

$$p(\boldsymbol{y}|\boldsymbol{z}) = \prod_i NB(\boldsymbol{y}_i|\boldsymbol{\mu}_i, \boldsymbol{\theta}).$$

Here, the NB likelihood of $y_{i,g}$ is calculated as

$$NB(y_{i,g}|\mu_{i,g},\theta_g) = \frac{\Gamma(y_{i,g}+\theta_g)}{y_{i,g}!\Gamma(\theta_g)}\left(\frac{\theta_g}{\theta_g+\mu_{i,g}}\right)^{\theta_g}\left(\frac{\mu_{i,g}}{\theta_g+\mu_{i,g}}\right)^{y_{i,g}},$$

where i represents the spot index and g represents the gene index. The NB parameter mean $\boldsymbol{\mu}$ is parameterized by decoder networks with respect to the latent embedding $\boldsymbol{z}$. Specifically,

$$\boldsymbol{\mu}_i = diag(s_i) \times \exp(l_{\boldsymbol{\mu}}(l(\boldsymbol{z})))$$

where $l_{\boldsymbol{\mu}}(\cdot)$ is a neural network that parametrizes the mean parameter, $l(\cdot)$ is the latent decoder for genes, and s_i is the library size factor of spot i that is calculated in the preprocessing step. An exponential activation function is appended to the network, given that the mean is always positive. In the model, the estimated mean can be used as denoised counts. The NB parameter dispersion θ_g for gene g is a trainable parameter (an exponential activation function is also applied to ensure that it is always positive).

The reconstruction loss of spot i's gene counts is

$$L_{gene} = -\log\left(NB(\boldsymbol{y}_i|\boldsymbol{\mu}_i,\boldsymbol{\theta})\right). \tag{2}$$

Latent space distribution. Priors

In the same way as the classical VAE, spaVAE estimates the latent mean and the latent variance by the encoder networks

$$\tilde{\boldsymbol{\omega}} = f_{\boldsymbol{\omega}}(f(\boldsymbol{y})),$$

$$\tilde{\boldsymbol{\phi}}^{\boldsymbol{2}} = \exp(f_{\boldsymbol{\phi}}(f(\boldsymbol{y}))),$$

where $f_{\boldsymbol{\omega}}(\cdot), f_{\boldsymbol{\phi}}(\cdot)$ are two neural networks that parameterize mean and variance, $f(\cdot)$ is the latent encoder, and $\tilde{q}(\tilde{\boldsymbol{z}}|\boldsymbol{y}) = \mathcal{N}(\tilde{\boldsymbol{\omega}}, \tilde{\boldsymbol{\phi}}^{\boldsymbol{2}})$. Here, the exponential activation function is used for the variance, because it is always positive.

The Gaussian process regression is applied to the first L dimensions of latent mean $\tilde{\boldsymbol{\omega}}$ and the latent variance $\tilde{\boldsymbol{\phi}}^{\mathbf{2}}$ to characterize the spatial dependency (denoted as $\tilde{\boldsymbol{\omega}}^{1:L}$ and $\tilde{\boldsymbol{\phi}}^{\mathbf{2^{1:L}}}$), and the other $D-L$ dimensions of $\tilde{\boldsymbol{\omega}}$ and $\tilde{\boldsymbol{\phi}}^{\mathbf{2}}$ follow a standard Gaussian prior (denoted as $\tilde{\boldsymbol{\omega}}^{L+1:D}$ and $\tilde{\boldsymbol{\phi}}^{\mathbf{2^{L+1:D}}}$).

As a result, the KL loss contains two components: the Gaussian process and Gaussian, and the ELBO can be written as

$$\begin{aligned} ELBO = {} & \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}[\log p(\boldsymbol{y}|\boldsymbol{z})] - \\ & \beta\left[KL\left(q(\boldsymbol{z}^{\mathbf{1:L}}|\boldsymbol{x},\boldsymbol{y})\|p(\boldsymbol{z}^{\mathbf{1:L}})\right) + KL\left(q(\boldsymbol{z}^{\boldsymbol{L+1:D}}|\boldsymbol{y})\|p(\boldsymbol{z}^{\boldsymbol{L+1:D}})\right)\right]. \end{aligned} \tag{3}$$

Gaussian Process prior

The prior distribution of the Gaussian process (GP) latent embedding $\boldsymbol{z}^{\mathbf{1:L}}$ is

$$p(\boldsymbol{z}^{\mathbf{1:L}}|\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, \boldsymbol{K_{NN}}),$$

where $\boldsymbol{K_{NN}}$ is the covariance matrix of the GP prior defined by a kernel function:

$$\boldsymbol{K_{NN}} = k_\theta(\boldsymbol{x}, \boldsymbol{x}),$$

and k_θ is the kernel function. If setting $\boldsymbol{K_{NN}} = \boldsymbol{I}$, then it will reduce to a traditional VAE. Different kernel functions can be used in this model to capture different spatial dependencies. Two kernel functions have been implemented: Cauchy and EQ (Gaussian) kernels. The authors have shown, spaVAE does not appear to be very sensitive to the choice of these kernels, while the Cauchy and EQ achieve the highest clustering accuracy.

The GP embedding part can be considered as a form of GP-VAE. Given the outputs of encoder $\tilde{\boldsymbol{z}}^{\mathbf{1:L}}$, GP-VAE combines latent representation learning and GP regression to generate $\boldsymbol{z}^{\mathbf{1:L}}$. Following Pearce (2020), the KL loss of GP embedding can be rewritten as

$$KL\left(q(\boldsymbol{z}^{\mathbf{1:L}}|\boldsymbol{x},\boldsymbol{y})\|p(\boldsymbol{z}^{\mathbf{1:L}})\right) = \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{z}^{\mathbf{1:L}}|\boldsymbol{x}_i,\boldsymbol{y}_i)}[\log \tilde{q}(\tilde{\boldsymbol{z}}^{\mathbf{1:L}}|\boldsymbol{y}_i)] - \sum_{l=1}^{L} \log Z^l(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N}), \tag{4}$$

where $\tilde{q}(\tilde{\boldsymbol{z}}^{\mathbf{1:L}}|\boldsymbol{y})$ and $q(\boldsymbol{z}^{\mathbf{1:L}}|\boldsymbol{x},\boldsymbol{y})$ represent the posterior distributions of the outputs of the encoder and GP regression, respectively. In the equation, i is the spot index, and $\boldsymbol{x}_{1:N}$ and $\boldsymbol{y}_{1:N}$ represent the whole dataset of N spots. The term $Z^l(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N})$ represents the marginal likelihood of the GP regression of the l-th latent dimension, and latent dimensions are assumed to be independent.

Sparse Gaussian Process regression

To accelerate the calculation and enable mini-batch stochastic optimization, the authors applied the sparse GP regression technique, which relies on inducing points. Denote the vector of p inducing points as $\boldsymbol{p}$. Given a set of inputs (spatial locations, inferred latent mean, and variance: $\boldsymbol{x}, \tilde{\boldsymbol{\omega}}, \tilde{\boldsymbol{\phi}}^{\mathbf{2}}$), following Hensman et al. (2013), then the sparse GP regression outputs $\boldsymbol{f_p} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{A})$. The ELBO L_H^l of the marginal likelihood of the sparse GP regression on the l-th latent dimension (in the first L dimensions) on the total N spots is

$$\begin{aligned} & Z^l(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N}) \geq L_H^l = \\ & \sum_{i=1}^{N}\left[\log \mathcal{N}(\tilde{\omega}_i^l | \boldsymbol{k}_i^T \boldsymbol{K_{pp}}^{-1} \boldsymbol{\mu}^l, \tilde{\phi}_i^{2^l}) - \frac{1}{\tilde{\phi}_i^{2^l}}\left(\tilde{k}_{ii} - trace(\boldsymbol{A}^l \Lambda_i)\right)\right] - KL(q_s^l(\boldsymbol{f_p})\|p(\boldsymbol{f_p})), \end{aligned} \tag{5}$$

where $\tilde{\omega}_i^l$ and $\tilde{\phi}_i^{2^l}$ represent the l-th latent dimension of the latent mean $\tilde{\boldsymbol{\omega}}$ and the latent variance $\tilde{\boldsymbol{\phi}}^{\mathbf{2}}$ for spot i respectively; $\boldsymbol{\mu}^l$ and $\boldsymbol{A}^l$ are the trainable parameters $\boldsymbol{\mu}$ and $\boldsymbol{A}$ for the

l-th latent dimension; $q_s^l(\boldsymbol{f_p}) = \mathcal{N}(\boldsymbol{f_p}|\boldsymbol{\mu}^l, \boldsymbol{A}^l)$ and $p(\boldsymbol{f_p}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K_{pp}})$ .Here, $\boldsymbol{K_{pp}} = k_\theta(\boldsymbol{p}, \boldsymbol{p})$, $\boldsymbol{k}_i$ represents the i-th column of $\boldsymbol{K_{pN}} = k_\theta(\boldsymbol{p}, \boldsymbol{x_N}) = \boldsymbol{K_{Np}}^T$, $\Lambda_i = \boldsymbol{K_{pp}}^{-1}\boldsymbol{k}_i\boldsymbol{k}_i^T\boldsymbol{K_{pp}}^{-1}$, $\tilde{k}_{ii}$ is the i-th diagonal element of $\boldsymbol{K_{NN}} - \boldsymbol{K_{Np}}\boldsymbol{K_{pp}}^{-1}\boldsymbol{K_{pN}}$. Following Titsias (2009); Jazbec et al. (2021), the stochastic estimates of $\boldsymbol{\mu}$ and $\boldsymbol{A}$ for each latent dimension l of a mini-batch b of data can be written:

$$\begin{aligned}
\Sigma_b^l &= \boldsymbol{K_{pp}} + \frac{N}{b}\boldsymbol{K_{pb}}diag(\tilde{\boldsymbol{\phi}}_{\boldsymbol{b}}^{\boldsymbol{-2^l}})\boldsymbol{K_{bp}}, \\
\boldsymbol{\mu}_b^l &= \frac{N}{b}\boldsymbol{K_{pp}}\left(\Sigma_b^l\right)^{-1}\boldsymbol{K_{pb}}diag(\tilde{\boldsymbol{\phi}}_{\boldsymbol{b}}^{\boldsymbol{-2^l}})\tilde{\boldsymbol{\omega}}_b^l, \\
\boldsymbol{A}_b^l &= \boldsymbol{K_{pp}}\left(\Sigma_b^l\right)^{-1}\boldsymbol{K_{pp}}.
\end{aligned} \tag{6}$$

The total GP regression ELBO combines L latent dimensions $L_H = \sum_{l=1}^{L} L_H^l$, and the stochastic GP regression ELBO on a mini-batch b of data can be obtained based on $\boldsymbol{\mu}_b$ and $\boldsymbol{A}_b$.

Posterior of Gaussian Process regression

Following Hensman et al. (2013), for the l-th latent dimension, the posterior distribution of the total N spots:

$$\begin{aligned}
q(\boldsymbol{z}^l|\boldsymbol{x}, \boldsymbol{y}) = q(\boldsymbol{z}_{1:N}^l) = \\
\mathcal{N}(\boldsymbol{z}_{1:N}^l|\boldsymbol{K_{Np}}\boldsymbol{K_{pp}}^{-1}\boldsymbol{\mu}^l, \boldsymbol{K_{NN}} - \boldsymbol{K_{Np}}\boldsymbol{K_{pp}}^{-1}\boldsymbol{K_{pN}} + \boldsymbol{K_{Np}}\boldsymbol{K_{pp}}^{-1}\boldsymbol{A}^l\boldsymbol{K_{pp}}^{-1}\boldsymbol{K_{pN}}).
\end{aligned} \tag{7}$$

And thus, given a mini-batch b of data, the stochastic estimate of the posterior distribution of latent embedding $\boldsymbol{z}^l$ can be found as

$$\begin{aligned}
\boldsymbol{m}_b^l &= \frac{N}{b}\boldsymbol{K_{bp}}\left(\Sigma_b^l\right)^{-1}\boldsymbol{K_{pb}}diag(\tilde{\boldsymbol{\phi}}_{\boldsymbol{b}}^{\boldsymbol{-2^l}})\tilde{\boldsymbol{\omega}}_b^l, \\
\boldsymbol{B}_b^l &= diag\left(\boldsymbol{K_{bb}} - \boldsymbol{K_{bp}}\boldsymbol{K_{pp}}^{-1}\boldsymbol{K_{pb}} + \boldsymbol{K_{bp}}\left(\Sigma_b^l\right)^{-1}\boldsymbol{K_{pb}}\right).
\end{aligned} \tag{8}$$

Thus, by combining all L latent dimensions, the posterior of the GP latent embedding becomes

$$q(\boldsymbol{z}^{1:L}|\boldsymbol{x}, \boldsymbol{y}) = \prod_{l=1}^{L} q(\boldsymbol{z}^l|\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{m}, \boldsymbol{B}).$$

Cross-Entropy

The term $\mathbb{E}_{q(\boldsymbol{z}^{\boldsymbol{1:L}}|\boldsymbol{x},\boldsymbol{y})}[\log \tilde{q}(\tilde{\boldsymbol{z}}^{\boldsymbol{1:L}}|\boldsymbol{y})]$ can be found:

$$\mathbb{E}_{q(\boldsymbol{z}^{\boldsymbol{1:L}}|\boldsymbol{x},\boldsymbol{y})}[\log \tilde{q}(\tilde{\boldsymbol{z}}^{\boldsymbol{1:L}}|\boldsymbol{y})] = -CE\left[\mathcal{N}(\boldsymbol{m}, \boldsymbol{B})\|\mathcal{N}(\tilde{\boldsymbol{\omega}}^{\boldsymbol{1:L}}, \tilde{\boldsymbol{\phi}}^{\boldsymbol{2^{1:L}}})\right]. \tag{9}$$

Gaussian prior

Following Kingma and Welling (2022), the posterior of standard Gaussian embedding follows $q(\boldsymbol{z}^{\boldsymbol{L+1:D}}|\boldsymbol{y}) = q(\tilde{\boldsymbol{z}}^{\boldsymbol{L+1:D}}|\boldsymbol{y}) = \mathcal{N}(\tilde{\boldsymbol{\omega}}^{\boldsymbol{L+1:D}}, \tilde{\boldsymbol{\phi}}^{\boldsymbol{2^{L+1:D}}})$, and thus, the KL loss of the standard Gaussian embedding $KL(q(\boldsymbol{z}^{\boldsymbol{L+1:D}}|\boldsymbol{y})\|p(\boldsymbol{z}^{\boldsymbol{L+1:D}}))$ is

$$KL(q(\tilde{\boldsymbol{z}}^{\boldsymbol{L+1:D}}|\boldsymbol{y})\|\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})) = -\frac{1}{2}\sum_{l=L+1}^{D}\left[\log\tilde{\phi}^{2^l} - \tilde{\phi}^{2^l} - \tilde{\omega}^{2^l} + 1\right]. \tag{10}$$

Learning objective

By combining (1)-(10), the learning objective of spaVAE as minimizing the equation on a mini-batch b of data can be written:

$$L_{spaVAE} = \sum_{i=1}^{b} -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x}_i,\boldsymbol{y}_i)} \log\left(NB(\boldsymbol{y}_i|\boldsymbol{z},\boldsymbol{\theta})\right) +$$

$$\beta\left[-CE\left[\mathcal{N}(\boldsymbol{m},\boldsymbol{B})\|\mathcal{N}(\tilde{\boldsymbol{\omega}}^{\boldsymbol{1:L}},\tilde{\boldsymbol{\phi}}^{\boldsymbol{2^{1:L}}})\right] - \frac{b}{N}L_H + KL(q(\tilde{\boldsymbol{z}}^{\boldsymbol{L+1:D}}|\boldsymbol{y}_i)\|\mathcal{N}(\boldsymbol{0},\boldsymbol{I}))\right].$$

Additional noise

To make the latent embedding more robust, especially when training on small datasets, an optional argument is used, which introduces the denoising technique to the encoder network. The noise is incorporated into the GP embedding only. For a mini-batch input $\boldsymbol{y}_b$, the model estimates two sets of latent means and variances:

$$\tilde{\boldsymbol{\omega}}^{\boldsymbol{1:L}} = f_{\boldsymbol{\omega}}(f(\boldsymbol{y}))[:,1:L], \tilde{\boldsymbol{\phi}}^{\boldsymbol{2^{1:L}}} = \exp(f_{\boldsymbol{\phi}}(f(\boldsymbol{y})))[:,1:L],$$

and

$$\tilde{\boldsymbol{\omega}}'^{\boldsymbol{1:L}} = f_{\boldsymbol{\omega}}(f(\boldsymbol{y}+\gamma\cdot\boldsymbol{\delta}))[:,1:L], \tilde{\boldsymbol{\phi}}'^{\boldsymbol{2^{1:L}}} = \exp(f_{\boldsymbol{\phi}}(f(\boldsymbol{y}+\gamma\cdot\boldsymbol{\delta})))[:,1:L],$$

where $\boldsymbol{\delta} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$, and γ controls the intensity of the random Gaussian noise. By equation (8), we can estimate two posterior means of the GP regression: $\boldsymbol{m}$ and $\boldsymbol{m'}$. Then the denoising loss function can be written by minimizing the mean square error between the two posterior means:

$$L_{noise} = \sum_{i=1}^{b}(\boldsymbol{m}_i - \boldsymbol{m'}_i)^2.$$

Implementation

Structure

The structure of the project is as follows. The root directory contains directories ***examples*** and ***spaVAE***, *requirements.txt* file with the list of necessary packages, and this very file (*MATH5472-Final-project-Report.pdf*).

The directory ***examples*** provides three samples (151673, 151510, 151507) from the *Human Dorsolateral prefrontal cortex (Human DLPFC)* dataset together with the processed files of the raw datasets, configurations for training and testing the models, and additional utility files (such as a file for splitting the dataset into train/test subsets and a file storing a modified plotting function from the source code of *umap* package for 151507).

The directory ***spaVAE*** contains multiple sections:

- ***dataset***: this module handles preprocessing of the raw counts and storing the data as the *torch.utils.data.Dataset*
- ***model***: this module deals with the model on every level, starting from the **kernel** (Cauchy and EQ kernels are implemented), **encoder/decoder neural networks** (dense encoder/decoder architecture and convolutional encoder architecture are implemented), **dynamic control of $\boldsymbol{\beta}$** weight of KL loss, **sparse variational GP**, and ending with the **spaVAE model** itself
- ***trainer***: this module handles **training and validation**, including **logging** of the metrics/losses, learning rates, β; **visualizing** the process with the **TensorBoard**; **saving checkpoints** of the current states of the model
- ***utils***: this module defines some helpful procedures for the **initial setup**, **logging**, **visualization**, and **metrics calculation**

Additionally, as a result of training and testing the model with 151673, 151510, 151507 samples, the automatically generated directories ***experiments*** and ***tests*** can be found there.

Finally, a possible **logger configuration** (*logger_ config.json*) is given as an example, **three testing files** (*test_ Human_ DLPFC_ sample_ 151673.py, test_ Human_ DLPFC_ sample_ 151510.py, test_ Human_ DLPFC_ sample_ 151507.py*), and **the training file** (*train.py*) are given to run train the model and run tests.

How to run the model

To train the model and launch tests, one needs gene counts data (*.h5* format) and a configuration file (*config.json*) to define the model and dataset and specify the training/testing settings.

For **training**, please run the command

```
python train.py -c <your-config-file.json>

                -l <your-logger-config-file.json>
```

from the ***spaVAE*** directory. For the full list of the parameters, please refer to the *train.py*.

After training, you'll find the automatically generated ***experiments*** directory with the ***checkpoint*** and ***log*** directories. ***checkpoint*** stores the configuration of your model (*config.json*), all checkpoints (*checkpoint-epoch<N>.pth* with the frequency specified in the configuration), and the checkpoint with the best model (*model_ best.pth*). ***log*** stores the log of the training process (*log.log*) and the information for TensorBoard (*events.out.tfevents...*) to visualize. All these can be found in the project for 151673, 151510, 151507 samples, except intermediate checkpoints to save the space.

For **testing** with 151673, 151510, 151507 samples, please run the command

```
python test_Human_DLPFC_sample_<sample-id>.py
    -c "examples/Human_DLPFC/sample_<sample-id>/config.json"
    -l <your-logger-config-file.json>
```

from the ***spaVAE*** directory.

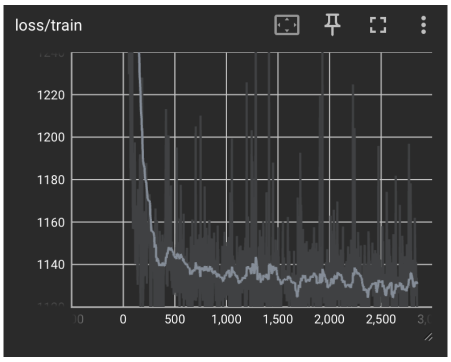
After testing, you'll find the automatically generated ***tests*** directory with the directories for each sample. Each of them has the configuration of your model (*config.json*), log of the testing process (*log.log*) and the information for TensorBoard visualization (*events.out.tfevents...*). Besides, the pictures of clustered spots' locations with respect to the latent representations, UMAP visualization of the latent space, and denoising/prediction of the counts can be found.

To **visualize** the training or testing process with **TensorBoard**, please run the command

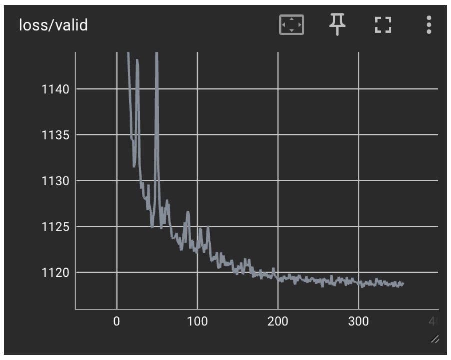
```
tensorboard --logdir=<path-to-log-dir/>
```

Here's an example of running TensorBoard with

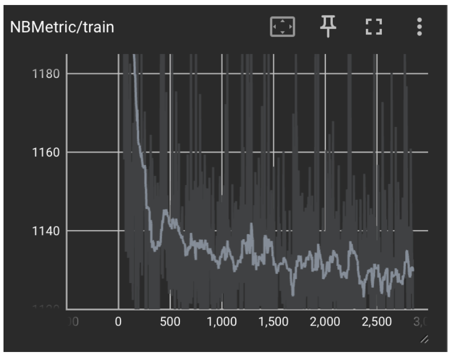
```
tensorboard logdir="experiments/Human_DLPFC_sample_151507/log"
```



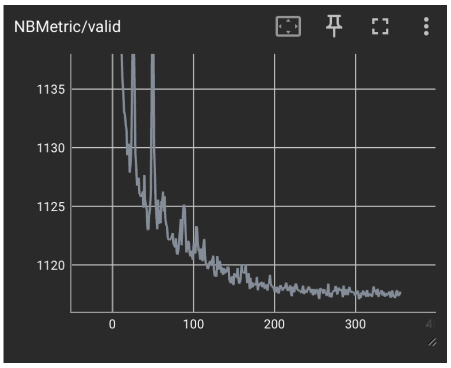
(a) Training total loss. Smoothened



(b) Validation total loss



(c) Training NB loss. Smoothened



(d) Validation NB loss

Figure 1: Training process. Total loss and NB loss for training batches and validation set. One training epoch = 8 steps

Source code

The project code, datasets, and results can be found here: `https://github.com/Chicky-Picky/Math5472-Final-project`.

Results

In this project, 151673, 151510, 151507 samples were used for training and testing.

Sample 151673

First, sample 151673 was picked to **reproduce the authors' results** in **clustering the spots' locations** with respect to the latent representations, **UMAP visualization of the latent space**, and **denoising the gene counts** (using the Ensembl gene to gene ID converter to identify gene IDs).

The model configuration:

- GP dimensions: 2
- Gaussian dimensions: 8
- Encoder hidden layers' dimensions: 128, 64
- Decoder hidden layer's dimension: 128
- Neural network type for the encoder: *Dense*
- Activation funciton: *ELU*
- Expected KL loss (for dynamic VAE): 0.025
- β_{min}: 4
- β_{init}: 10
- β_{max}: 25
- Kernel: Cauchy
- Initial kernel scale: 20
- Fixed inducing points: *True*
- Additional noise: 0.0

The training configuration:

- Validation split: 5%
- Batch size: 512
- Optimization algorithm: *AdamW*
- Initial learning rate: 0.1
- Weight decay: 0.000001
- Scheduler type: *Reduce on plateau*
  - Factor: 0.75
  - Patience: 15
- Maximum epochs number: 1500
- Checkpoint frequency: 50
- Early stop (number of epochs with no improvement on the validation set): 50

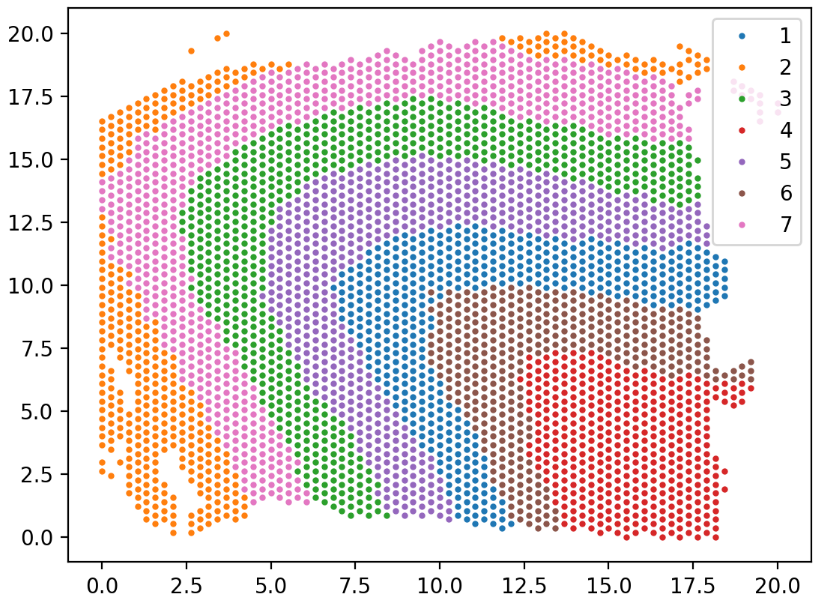


Figure 2: Clustered spots' locations. NMI = 0.712, ARI = 0.546.

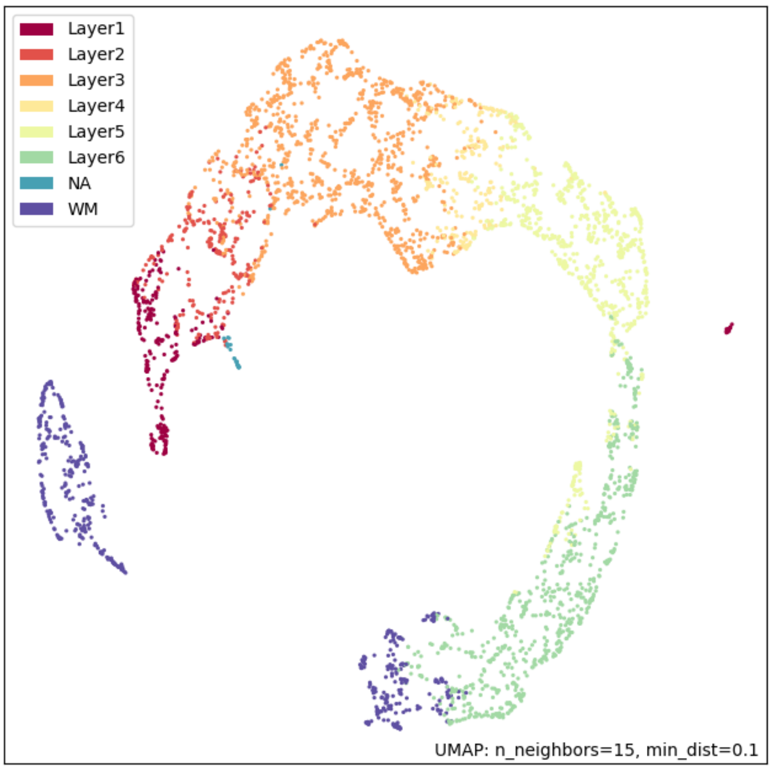


Figure 3: UMAP representation of the latent space with the ground true labels.

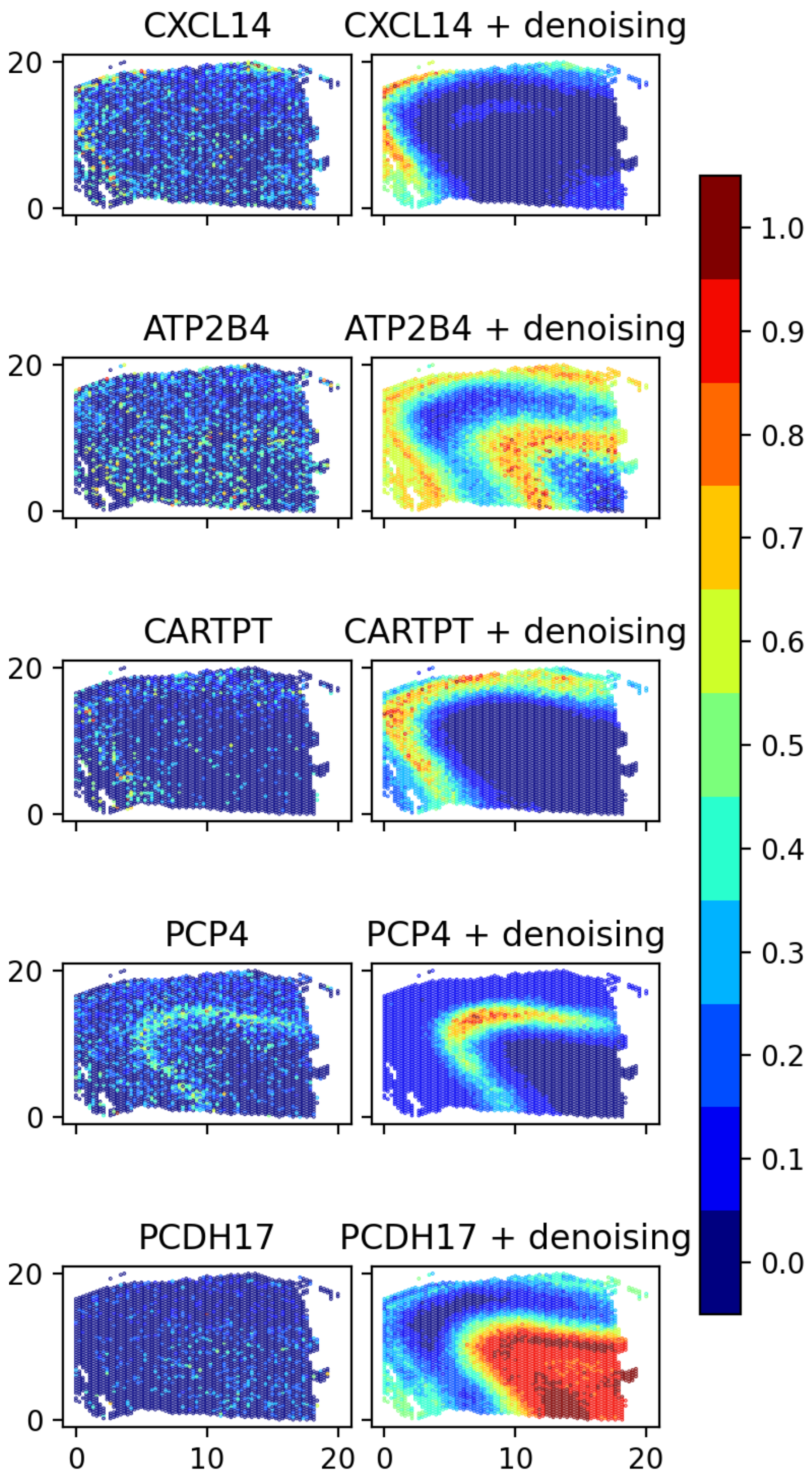


Figure 4: Initial and denoised counts.

Sample 151510

Then, sample 151510 was picked to test **the convolutional architecture** (with less parameters) and **EQ kernel** in **clustering the spots' locations** with respect to the latent representations, **UMAP visualization of the latent space**, and **denoising the gene counts** (using the Ensembl gene to gene ID converter to identify gene IDs).

The model configuration:

- GP dimensions: 2
- Gaussian dimensions: 8
- Encoder hidden layers' dimensions: 128, 64
- Decoder hidden layer's dimension: 128
- Neural network type for the encoder: *Convolutional*
- Activation funciton: *ELU*
- Expected KL loss (for dynamic VAE): 0.025
- β_{min}: 4
- β_{init}: 10
- β_{max}: 25
- Kernel: EQ
- Initial kernel scale: 20
- Fixed inducing points: *True*
- Additional noise: 0.0

The training configuration:

- Validation split: 5%
- Batch size: 512
- Optimization algorithm: *AdamW*
- Initial learning rate: 0.1
- Weight decay: 0.000001
- Scheduler type: *Reduce on plateau*
  - Factor: 0.75
  - Patience: 10
- Maximum epochs number: 1500
- Checkpoint frequency: 50
- Early stop (number of epochs with no improvement on the validation set): 50

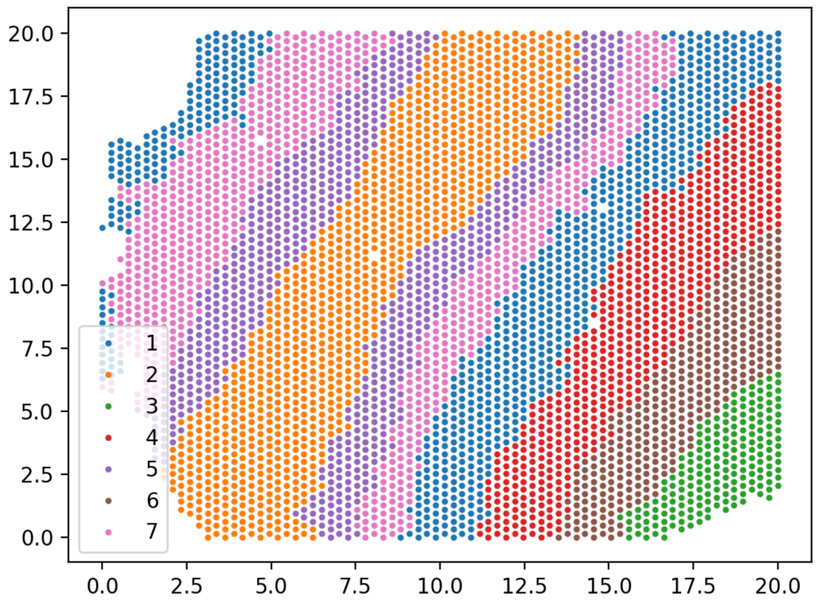


Figure 5: Clustered spots' locations. NMI = 0.651, ARI = 0.509.

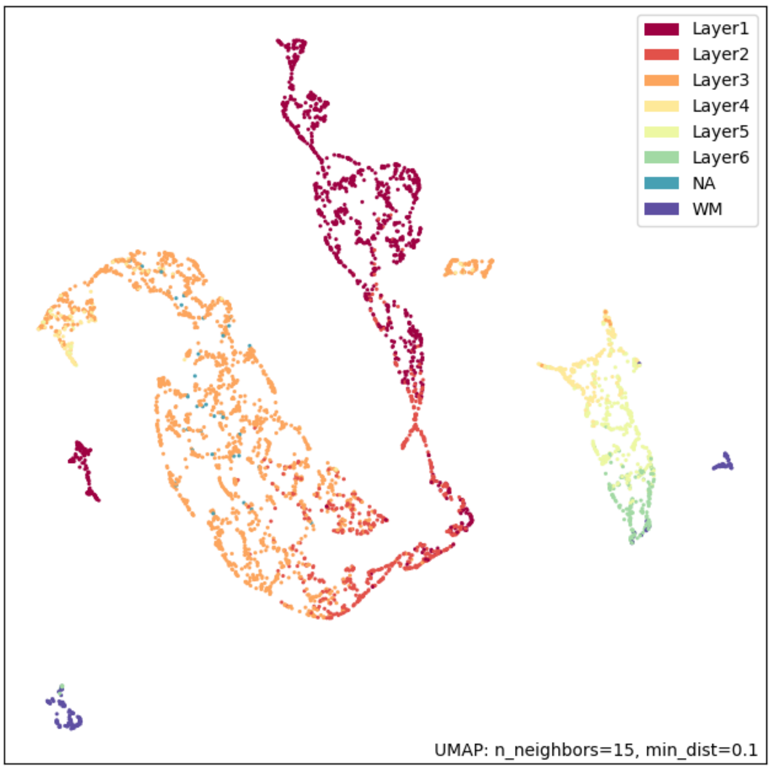


Figure 6: UMAP representation of the latent space with the ground true labels.

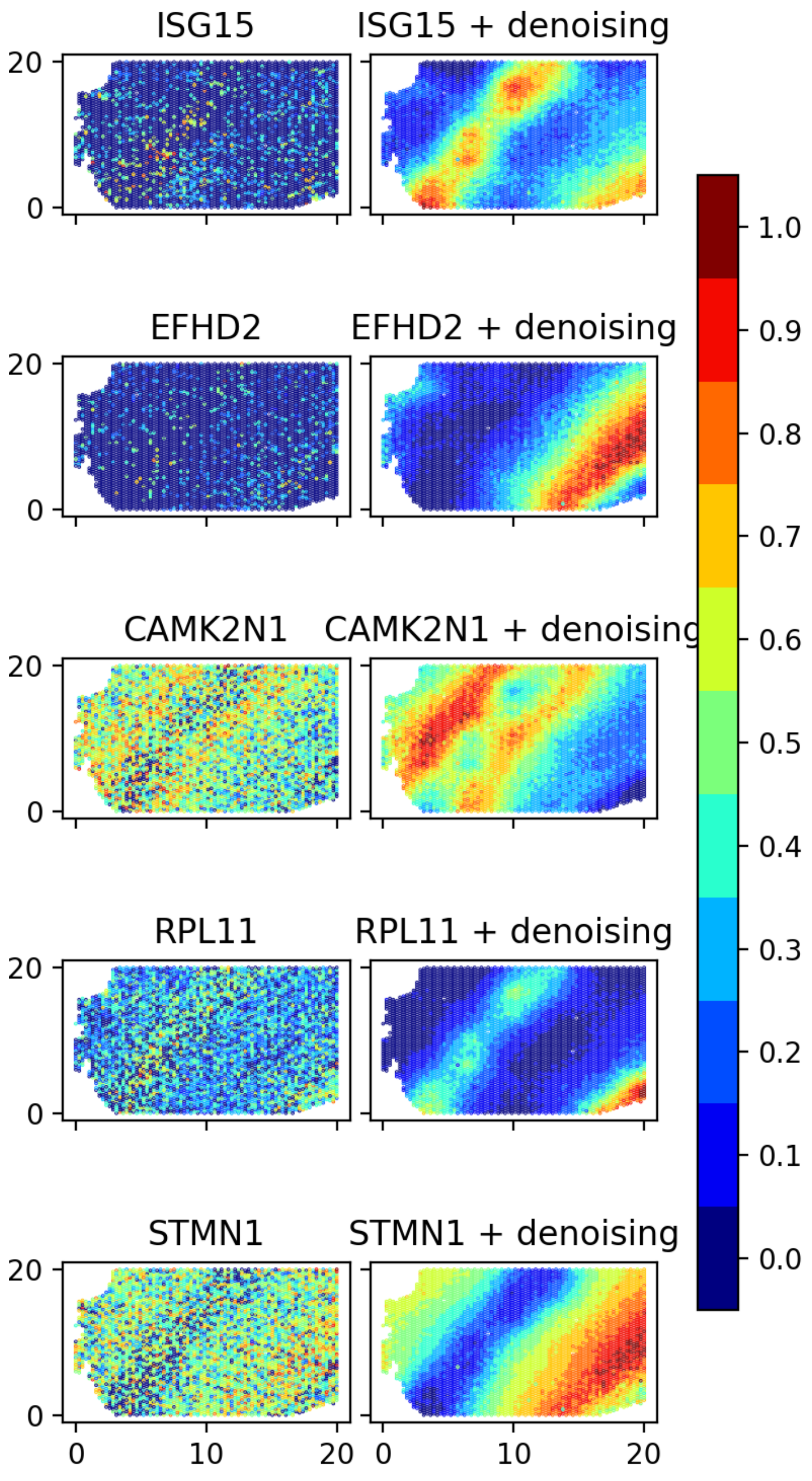


Figure 7: Initial and denoised counts.

Sample 151507

Finally, sample 151507 was picked to **test prediction of the gene counts**, besides **clustering the spots' locations** with respect to the latent representations, **UMAP visualization of the latent space**, and **denoising the gene counts** (using the Ensembl gene to gene ID converter to identify gene IDs).

The model configuration:

- GP dimensions: 2
- Gaussian dimensions: 8
- Encoder hidden layers' dimensions: 128, 64
- Decoder hidden layer's dimension: 128
- Neural network type for the encoder: *Dense*
- Activation funciton: *ELU*
- Expected KL loss (for dynamic VAE): 0.025
- β_{min}: 4
- β_{init}: 10
- β_{max}: 25
- Kernel: Cauchy
- Initial kernel scale: 20
- Fixed inducing points: *True*
- Additional noise: 0.0

The training configuration:

- Test split: 10%
- Validation split: 5%
- Batch size: 512
- Optimization algorithm: *AdamW*
- Initial learning rate: 0.1
- Weight decay: 0.000001
- Scheduler type: *Reduce on plateau*
  - Factor: 0.75
  - Patience: 15
- Maximum epochs number: 1500
- Checkpoint frequency: 50
- Early stop (number of epochs with no improvement on the validation set): 50

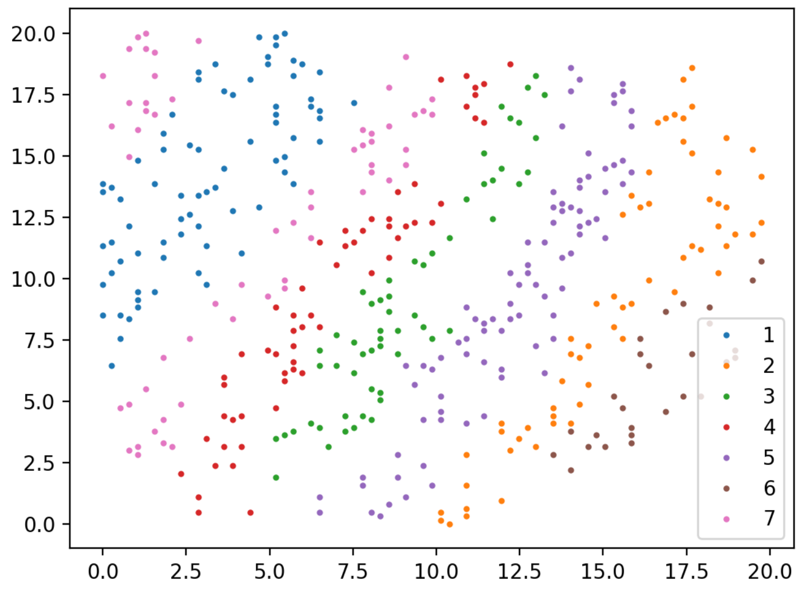


Figure 8: Clustered test spots' locations. NMI = 0.709, ARI = 0.558. Prediction accuracy: 0.887 (KNN with 11 neighbors).

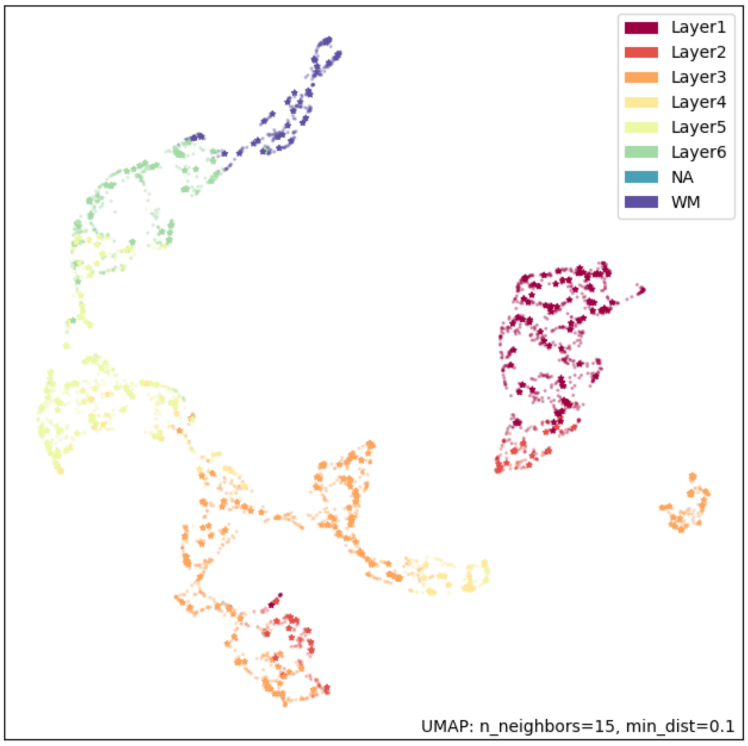


Figure 9: UMAP representation of the latent space with the ground true labels. Faded points — train set, star-shaped points — test set.

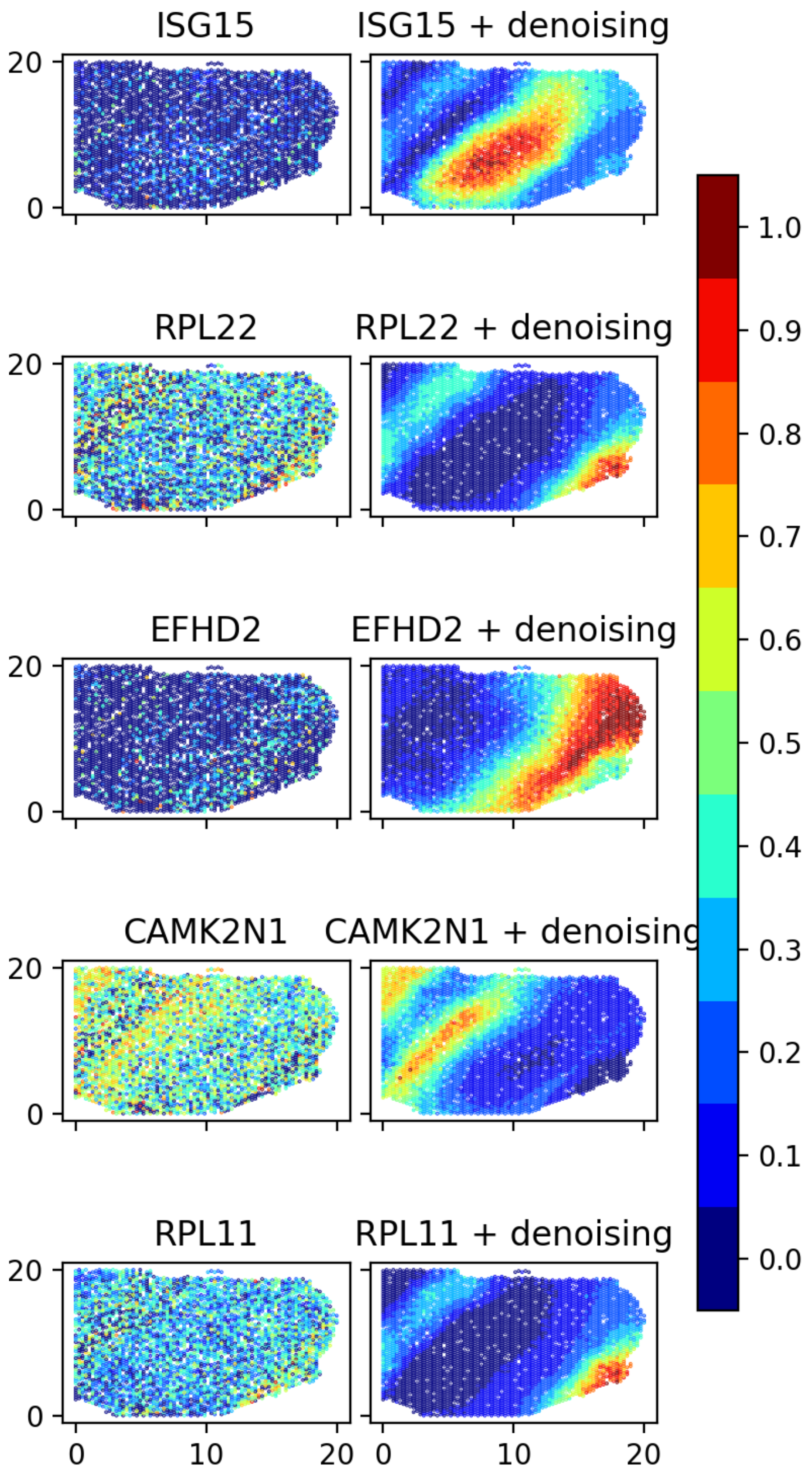


Figure 10: Initial train counts and denoised + predicted counts.

Discussion

spaVAE is a deep generative model designed to account for spatial dependency in spatial omics data while automatically optimizing the strength of this dependency from the data. This model directly handles discrete gene count data using model-based NB loss function and performs well across various spatial omics platforms and tissues. spaVAE supports key analytical tasks such as visualization, clustering, denoising (and, consequently, differential expression), resolution enhancement, and spatial interpolation, which are essential for uncovering biological insights like identifying new subpopulations and marker genes.

Variations and expansions

The authors also introduced multiple variations and expansions of spaVAE. For example, to identify spatially variable genes (SVG) **spaLDVAE** is used. The spaLDVAE model is based on spaVAE. For interpretability purposes, the authors suggest to use a linear decoder in spaLDVAE. Thus, for the SRT data, the mean parameter in the NB likelihood is estimated by

$$\boldsymbol{\mu}_i = diag(s_i) \times [\exp(\boldsymbol{z}_i) \times \exp(\boldsymbol{W}_D)],$$

where $\boldsymbol{z}$ is the latent embedding and $\boldsymbol{W}_D$ is the trainable decoder weight. Here, an exponential function is used to ensure that these two parameters are always positive.

A few other variations can be defined by introducing **new reconstruction losses** such as binary cross-entropy, likelihood of a mixture of two negative binomial distributions, depending on the task and data.

Additionally, to integrate datasets from different sources (**batch integration**), the authors use the conditional autoencoder technique Sohn et al. (2015) to learn a batch-free representation. Specifically, if we denote the encoder as f and decoder as l, then $\boldsymbol{z} = f(\boldsymbol{y}, \boldsymbol{x}_{batch})$ and $\boldsymbol{y'} = l(\boldsymbol{z}, \boldsymbol{x}_{batch})$, where $\boldsymbol{x}_{batch}$ is the one-hot encoded batch identification number (ID). The resulting $\boldsymbol{z}$ is expected to be disentangled from the batch IDs. For the GP part, a kernel needs to integrate batches. In this situation the auxiliary information $\boldsymbol{x}$ can be divided into two parts, one is the one-hot encoded batch ID $\boldsymbol{x}_{batch}$ and the other is the spatial location $\boldsymbol{x}_{spatial}$. The kernel function which integrates batches can be written as

$$k_\theta(\boldsymbol{x}, \boldsymbol{x}) = \boldsymbol{x}_{batch}\boldsymbol{x}_{batch}^T k_\theta(\boldsymbol{x}_{spatial}, \boldsymbol{x}_{spatial}).$$

This kernel function considers the independency between different batches, as well as the spatial dependency within one batch of data.

Disadvantages and issues

Despite its strengths and versatility, the model has several limitations that should be considered:

- **Dependency on inducing points:**
  - The models require users to manually specify the number of inducing points for the sparse Gaussian process regression. While inducing points reduce computational complexity, determining the optimal number is not automated and depends on the complexity of the spatial structure in the dataset
  - Complex spatial structures may require more inducing points, which increases computational demands
- **Challenges with small datasets:** The models are optimized for medium to large datasets and may struggle with small datasets (e.g., those with fewer than 500 spots). Training on such datasets can be challenging due to the limited data available for model optimization

- **Lack of integration with imaging data:** The models currently do not incorporate complementary information from microscopy imaging data, which is often generated alongside spatial omics data and could provide valuable context for analysis

My thoughts

I have always believed that VAEs are not only highly practical but also profoundly elegant in their design. As a variational Bayesian method, they offer a remarkably powerful and robust framework for statistical inference and prediction. In this paper, a seemingly straightforward generalization — introducing a Gaussian process (GP) prior alongside the Gaussian prior — enables the incorporation of spatial location information, which significantly enhances the performance of the models in the context of SRT data analysis.

The first thought that came to mind after reading the paper was: *What if we incorporate not only spatial but also temporal data?* Spatiotemporal omics appears to be an emerging frontier, particularly in precision medicine, as highlighted in Zhang et al. (2022); Liu et al. (2024). Interestingly, there are existing frameworks like Spatiotemporal Variational Gaussian Processes (e.g., Hamelijnck et al. (2021)), which could serve as a natural extension to spaVAE, opening the door to even more comprehensive modeling approaches.

I was also curious about a practical question: *How can users determine the optimal latent space dimension for their data?* While one could test multiple configurations and compare their performance, this approach is resource-intensive and time-consuming. This remains an active area of research, but promising solutions have been proposed, such as in Sejnova et al. (2023), which could potentially be integrated into spaVAE to improve computational efficiency and ease of use.

This paper has sparked a deep interest in me to explore VAEs further. Their potential for innovation, particularly with spatiotemporal extensions and optimization strategies, is immense. I look forward to diving deeper into this topic and uncovering more about the exciting possibilities of VAEs in omics and beyond!

References

Hamelijnck, O., Wilkinson, W. J., Loppi, N. A., Solin, A., and Damoulas, T. (2021). Spatio-temporal variational gaussian processes. *CoRR*, abs/2111.01732.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data.

Jazbec, M., Ashman, M., Fortuin, V., Pearce, M., Mandt, S., and Rätsch, G. (2021). Scalable gaussian process variational autoencoders.

Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes.

Liu, L., Chen, A., Li, Y., Mulder, J., Heyn, H., and Xu, X. (2024). Spatiotemporal omics for biology and medicine. *Cell*, 187(17):4488–4519.

Pearce, M. (2020). The gaussian process prior vae for interpretable latent dynamics from pixels. In Zhang, C., Ruiz, F., Bui, T., Dieng, A. B., and Liang, D., editors, *Proceedings of The 2nd Symposium on Advances in Approximate Bayesian Inference*, volume 118 of *Proceedings of Machine Learning Research*, pages 1–12. PMLR.

Sejnova, G., Vavrecka, M., and Stepanova, K. (2023). Adaptive compression of the latent space in variational autoencoders.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *Neural Information Processing Systems*.

Tian, T., Zhang, J., Lin, X., Wei, Z., and Hakonarson, H. (2024). Dependency-aware deep generative models for multitasking analysis of spatial omics data. *Nature Methods*, 21(8):1501–1513.

Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.

Zhang, J., Yin, J., Heng, Y., Xie, K., Chen, A., Amit, I., Bian, X.-w., and Xu, X. (2022). Spatiotemporal omics-refining the landscape of precision medicine. *Life Medicine*, 1(2):84–102.