Assignment 3:
Advanced topics: dimensionality reduction (standard and feature extraction based), visualization, evaluation

Scientific Visualization 2021/22 (WMCS018-05.2021-2022.1B)
v1.1

January 13, 2022

Submission Deadline: January 24 (2022), 10:59am

1 General Information

The assignment is designed to be addressed alongside the lectures, with the individual tasks covering what has been discussed in the respective week. The tasks may also be tackled in advance, and for this the most essential information (or respective links) are provided as part of this assignment. For a brief introduction to the code structure and instructions how to get started, please refer to the notes of the warm-up assignment.

Submission. For each assignment you have to hand in an archive including the source code and your report via the dropbox on Nestor. It needs to be submitted before the respective deadline, otherwise a penalty is applied. Please follow the guidelines on reports as listed on the Nestor page.

2 Tasks

Prerequisites: obtain the dataset (Kármán Vortex Street ensemble) via Dropbox: `https://www.dropbox.com/sh/qsoki0i5f76umf5/AACkkbTj-qaxNgOUwjgOx5A2a?dl=0`

About the dataset: Kármán Vortex Street is an ensemble of 300 fluid flow simulations (members) with 54 timesteps each, containing flow in a "tube" obstructed by a cylindrical obstacle. The timesteps (images) are monochrome and have a resolution of 441×84. Depending on it's size, position and fluid viscosity the flow can be very

turbulent or completely laminar, as depicted in the examples. In the beginning of each member, there are several time steps which are empty, these can be neglected. The names of the members contain three simulation parameters: Reynold's number, cylinder offset, and cylinder radius. For instance, `cylinder_100_5_30` means the Reynold's number is 100, cylinder offset is 5 and radius is 30.

The data is in Datraw format, essentially each volume (simulation) is a '.dat' metadata file and a folder of binary files, one for each timestep. Individual timesteps can be loaded just using Numpy's 'numpy.fromfile', and then reshaped according to the resolution (specified in the metadata). All data is UCHAR, i.e. `numpy.uint8`. In order to save time and not to load the raw data each time, you can save all data (e.g. as a pickle using `pickle.dump`) once and then load from there.

The example below shows a few rendered timesteps from different simulations, you can use it to check what correctly loaded data looks like (here the data is visualized using 'viridis' colormap from PyPlot).

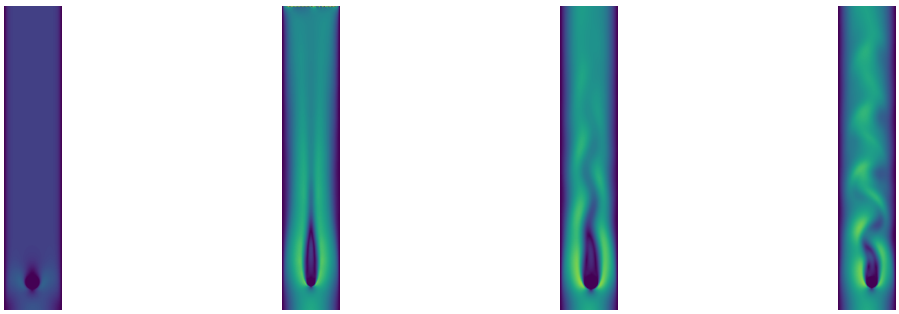

Figure 1: Examples of the Kármán Vortex Street frames.

Task 1 – Data loading and pre-processing (2 points)

Obtain the Python code `unlabeled_data_pickle_flow.py` and `labeled_data_pickle_flow.py` (both use `feature-metadata-vortex.json` file) for loading the Vortex Street unlabeled and labeled data. There are 16200 timesteps in total, you are not required to use the full dataset (the code loads a subset). Load the dataset and apply standard preprocessing steps such as normalization (to zero mean and unit variance) and cropping. You can use the `preprocessing.py` script where preprocessing and cropping is implemented, and apply these steps to the generated `.pkl` data files. Since the images from this dataset contain regions that are irrelevant for the following tasks (projections), it is important to remove them during preprocessing (cropping). Visualize timesteps of the loaded dataset using e.g. 'viridis' colormap. For instance, the cropped frames may look as depicted in Figure 2.

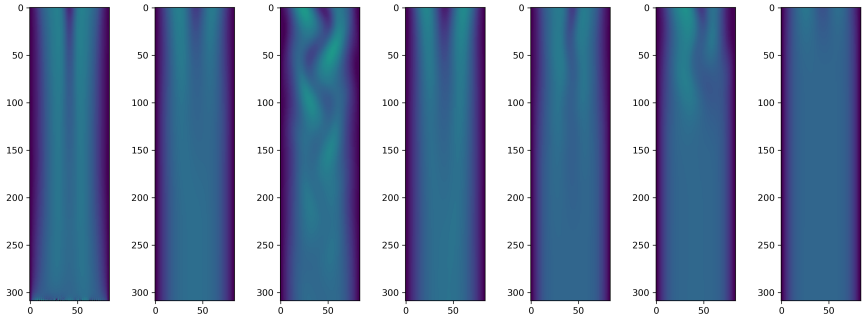

Figure 2: Vortex Street, examples of the cropped timesteps.

Task 2 – Standard dimensionality reduction, visualization, and evaluation (4 points)

Apply the standard dimensionality reduction (DR) techniques, such as PCA, t-SNE, and UMAP, to the preprocessed data. Reduce the dimensionality of the data from original frames size to 2D. You can use scikit-learn library for that (`https://scikit-learn.org/stable/`). Visualize the results in a form a 2D scatterplot (Figure 3, right) and also in such a way that for each data point on the reduced 2D map, you see the corresponding image (Figure 3, left). You can use matplotlib library for that (`https://matplotlib.org/`), as well as the `scatterplot_with_imgs()` function from the `img_scatterplot.py` file. This function receives two dimensional components (x and y) of the data points to be visualized, `data` matrix containing the images to be places instead of the points, `ax` from `plt.subplots()`, as well as `zoom` variable that controls the visualized images size. You can also try plotting the data as individual points, distinguishing them via e.g. shape, hue, or color.

Implement the *neighborhood hit* metric [2] and evaluate the obtained results. This metric is based on k-nearest neighbors and computes the fraction of neighbors belonging to the same class for each labeled data point. The value that each point contributes to the total result is fractional, and it provides an accurate measure of the clusters separation. The output is in the range of [0, 1], where *higher values* represent a better separation of the clusters. You can set the k to 17 for the implementation. In order to perform evaluation with this metric, you need to utilize the labeled subset.

Example of the 2D projected data with UMAP is shown in Figure 3. The left side of the figure shows a projection with the corresponding frames and the right is in a form of a scatterplot.

Task 3 – Feature extraction based dimensionality reduction (6 points)

Now implement a simple autoencoder (AE) so that to compress the input data and learn the most import features necessary for the dimensionality reduction. Use the extracted

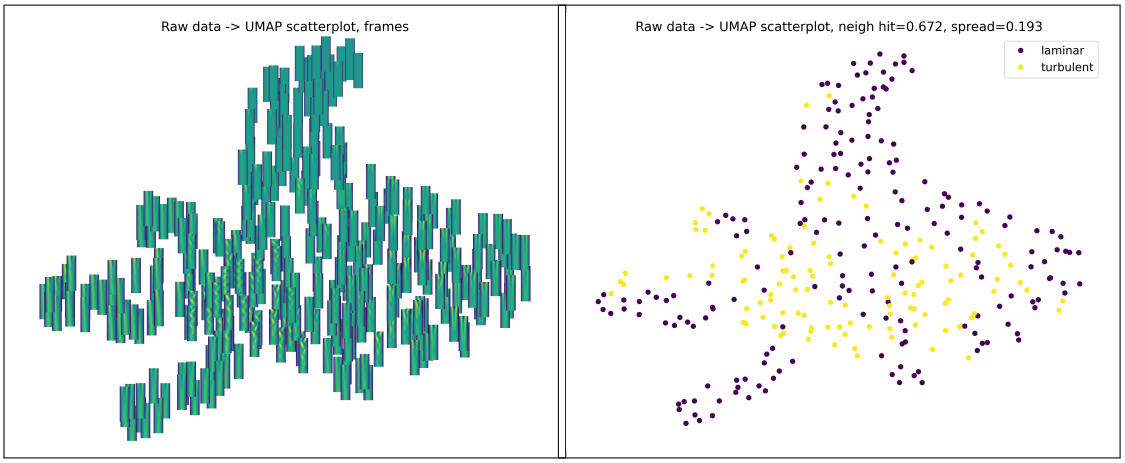

Figure 3: DR results on Vortex Street ensemble dataset: left - direct UMAP projection shown with time steps, right - corresponding UMAP scatterplot.

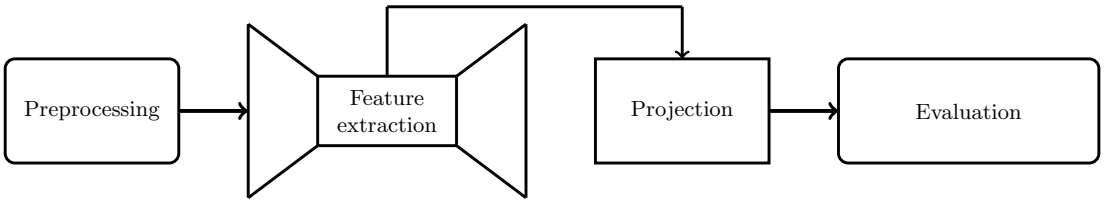

Figure 4: Overview: start by preprocessing the data which is then applied (left trapezoid part) to an autoencoder-based feature extraction method. An autoencoder then reconstructs the data (right trapezoid part), however, we are interested in the extracted features (middle part) which are then used for the projection techniques. After obtaining the visualization, it is further evaluated with the metric.

features (in the form of latent vectors, see below) to further reduce the dimensions to 2D with the standard dimensionality reduction techniques or reduce them directly to 2D using autoencoder. You may refer to the `Autoencoders for Dimensionality Reduction.pdf` paper where autoencoders were utilized, including the simplest standard version. Figure 4 shows the pipeline of this task.

Autoencoder hyper-parameters. You may implement an autoencoder with 4 convolutional layers in the encoding. To reduce the resolution of the input image by half on each convolutional layer, you can use the parameter *stride* of 2 for each dimension. The *kernel size* for convolution operations can be set to 3 for each dimension. The *number of filters* parameter in each layer can be set to 64. The parameter *padding* can be specified as "same" for zero padding.

Apply the fully connected (dense) layer after the group of convolutional layers. Re-

shape the tensors to flatten them before that. The dimension of the fully connected layer, which follows the convolutions, can be set to be greater than the dimension of the latent space. The vectors from this latent space should be applied to the standard DR techniques in order to achieve a 2D projection (if their dimension in greater than 2). For instance, if the dimension of latent space is 256, then the previous fully connected layer could have 1024 units. Note that these dimensions should be selected so that to keep proportion of the compression rate at each layer.

After that, the second dense layer generates the latent representation (latent vectors, or embedding). This is the layer that has the smallest dimensionality in the network and therefore is also known as the "bottleneck". You may perform experiments with different latent space dimensions, from 2 to 512.

In order to prevent overfitting, you may apply regularizations (L1 and L2) to the dense layer that produced the latent space. That will make the autoencoder sparse.

The autoencoder's decoder is symmetric to the encoder. The first layer of the decoder can be a fully connected dense layer that upsamples the input to be suitable for convolutional layers. After reshaping the vectors to tensors, 4 transposed convolutional (deconvolutional) layers can be followed. In the output, a linear activation can be used.

Train the autoencoder (10-20 epochs should be sufficient) e.g. using Adaptive Moment Estimation (Adam) optimizer with the parameter *learning rate* of 0.0005 and Mean Squared Error (MSE) loss. Check the reconstruction results after training. A properly trained autoencoder should be able to reconstruct the input images almost perfectly (e.g. as in Figure 5, with 10 training epochs). Using 2000 unlabeled images for training and 300 labeled images for inference is sufficient to demonstrate the concept. Visualize the results in a similar way as for the task 2. Compare the results with those obtained using standard DR techniques on the raw preprocessed data (without autoencoder). Do you see any improvement in terms of metric score?

For instance, the result using an autoencoder could look as depicted in Figure 6, All laminar data points are shown in purple and turbulent ones in yellow in the scatterplot projection on the right.

You can take a look at the paper where Kármán dataset was used: `https://diglib.eg.org/handle/10.2312/stag20191367`. All these steps were described in the paper: `Autoencoders for Dimensionality Reduction.pdf` [1]. You can use Keras as a machine learning framework and refer to the tutorial: `https://blog.keras.io/building-autoencoders-in-keras.html`. Of course, TensorFlow or PyTorch can be used as well.

Task 4 – Bonus: analyze a different dataset (8 points)

Note that this is a bonus task - you will not lose any points if you don't have time to implement this task.

Obtain the second dataset via Dropbox: `https://www.dropbox.com/s/mskhs1mp6xmjah8/mcmc_a.raw.bz2?dl=0`.

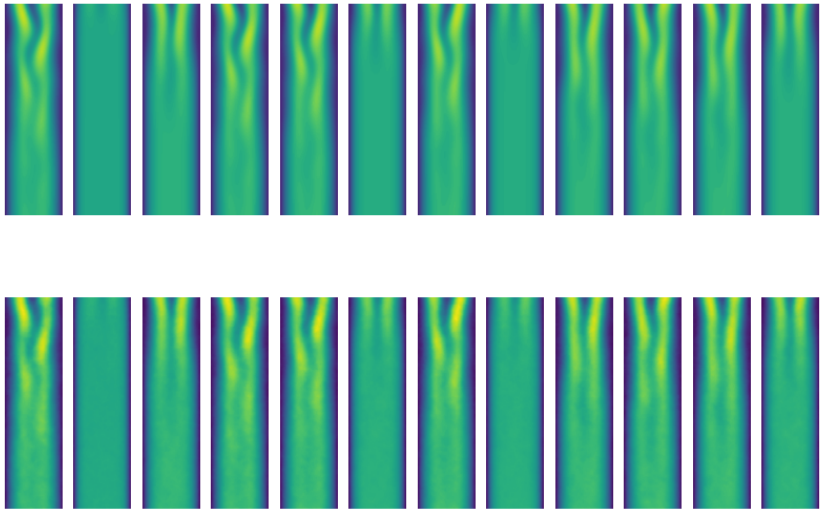

Figure 5: Vortex Street, examples of the input (top) and reconstructed (bottom) images.

This ensemble stems from a Markov chain Monte Carlo (MCMC) process carrying out inverse calculations of channel structures in the soil from sparse, non-transient hydraulic head measurement data. Each sample/member consists of a grid of scalar values that represent the estimated hydraulic conductivity of the soil. The data is compressed via bzip2 and contains 95000 members, each of which is 50×50 pixels, whereas each pixel contains a 64 bit floating point value. The data is stored in one raw file in the order of (x, y, Members), i.e, the first 50 elements depict the first row (the x-dimension) of the first member, and the first 2500 elements describe the first member.

Members should approximately look like as depicted here: `https://freysn.github.io/wasm/hgl-demo/mcmc_95k_128k.html`. Some other examples are in Figure 7.

Repeat all steps from previous tasks using MCMC dataset. For loading the MCMC unlabeled and labeled data, use the Python code `from_raw_to_pickle_mcmc.py` (uses `labels_mcmc.txt` file). There, the boolean `labeled` controls whether the labeled only subset or the whole dataset will be saved. Note that cropping is not necessary for this dataset since the images do not contain irrelevant regions. Using around 5000 unlabeled images for training and 1000 labeled images for inference is sufficient to demonstrate the concept. The labeled MCMC subset consists of five categorical classes (● ● ● ● ●) which depict qualitatively different types of channel structures. You may compare your results with those shown in Figure 8 and Figure 9.

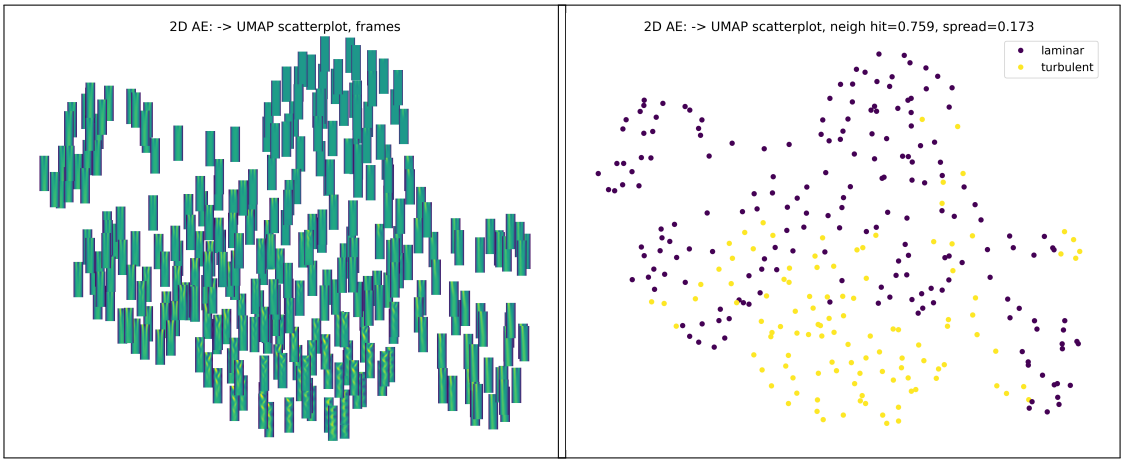

Figure 6: DR using 2D Sparse AE with latent space dim. of 128: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

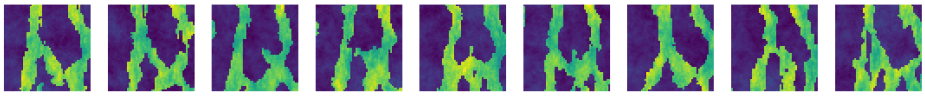

Figure 7: Sample images from MCMC simulation

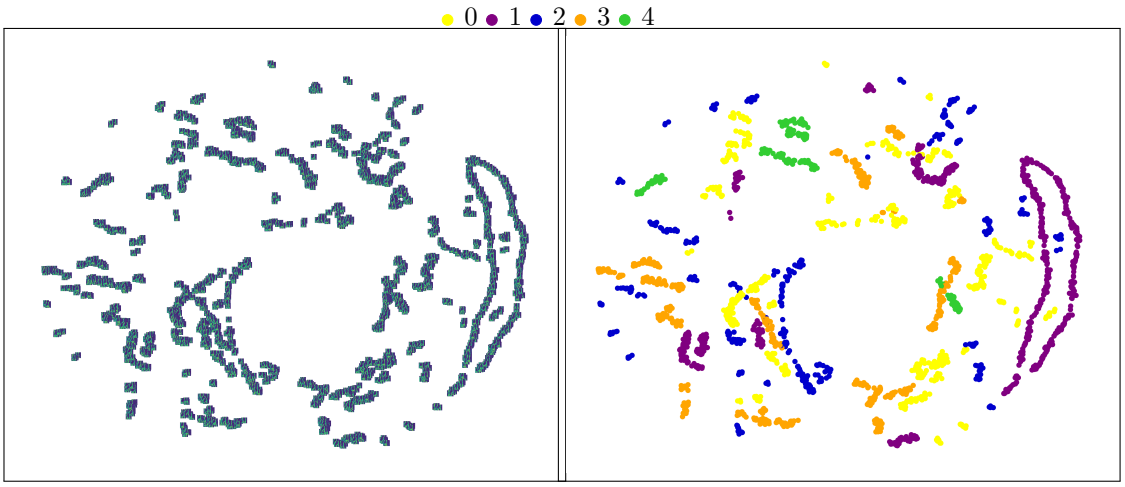

Figure 8: DR results on MCMC ensemble dataset: left - direct UMAP projection shown with time steps, right - corresponding UMAP scatterplot.

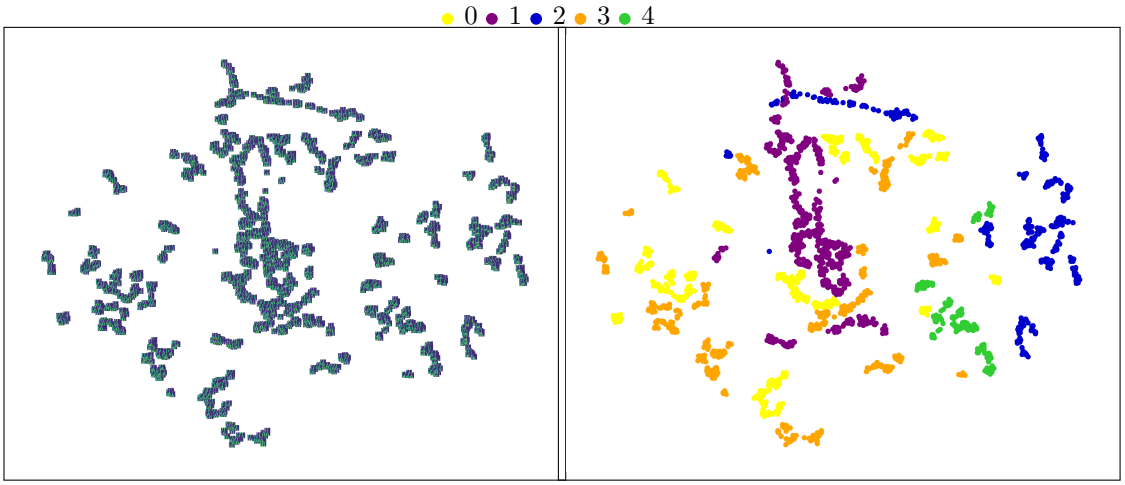

Figure 9: AE based feature extraction results on MCMC ensemble dataset: left - direct UMAP projection shown with time steps, right - corresponding UMAP scatterplot.

References

[1] H. Gadirov, G. Tkachev, T. Ertl, and S. Frey. Evaluation and selection of autoencoders for expressive dimensionality reduction of spatial ensembles. In *International Symposium on Visual Computing*, pages 222–234. Springer, 2021.

[2] E. F. Vernier, R. Garcia, I. da Silva, J. L. D. Comba, and A. C. Telea. Quantitative evaluation of time-dependent multidimensional projection techniques. *arXiv preprint arXiv:2002.07481*, 2020.