10907 Pattern Recognition

Lecturers
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩

Tutors
Alexandra Flora Spitzer ⟨alexandra.spitzer@stud.unibas.ch⟩
Roman Fries ⟨r.fries@unibas.ch⟩
Cheng Shi ⟨cheng.shi@unibas.ch⟩
Vinith Kishore ⟨vinith.kishore@unibas.ch⟩
Valentin Debarnot ⟨valentin.debarnot@unibas.ch⟩

Assignment 6

Deadline: 21.12.2023
Total points: 6

Academic integrity You are encouraged to discuss the material with others, but it is essential to document these discussions by writing down the names of the individuals you worked with and any tools that helped you with the assignment. Any instance of cheating will be dealt extremely strictly, potentially resulting in receiving zero credit for the course at a minimum.

FAQ

1. What should I do after uploading my code? Double or triple check! make sure it goes through all the test cases and a score appears. This score is not your final score because you can correct the reported errors and resubmit the code again for a higher score. This is why I highly encourage uploading your code early so there is enough time to debug.
2. What do I do when seeing an error from Gradescope? Let us know as soon as possible. It could be that your implementation take too long to run (timeout error).
3. What to do if the TA makes a mistake grading my solution? If you disagree with the score, you can request a Regrade on Gradescope and we will take a second look as soon as possible.
4. How to use Piazza forum effectively? Open a public thread whenever you have a question regarding the class, homework, Gradescope, etc. You can choose to post anonymously. Some of us will try to answer quickly.

Math

Exercise 1 (Principal Component Analysis — 2 points).

Suppose we want to find an orthogonal set of p linear basis vectors $\boldsymbol{\phi}_j \in \mathbb{R}^d$, and the corresponding weights $\boldsymbol{w}_i \in \mathbb{R}^p$ for n data points $\boldsymbol{x}_i \in \mathbb{R}^d$, such that we minimize the average reconstruction error

$$L(\boldsymbol{\Phi}, \boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2 \quad \text{where} \quad \tilde{\boldsymbol{x}}_i = \sum_j \boldsymbol{\phi}_j w_{ij}. \tag{1}$$

Here we denote $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_p]$ and $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]$. The optimal solution $L(\hat{\boldsymbol{\Phi}}, \hat{\boldsymbol{W}}) = \min L(\boldsymbol{\Phi}, \boldsymbol{W})$ is obtained by setting $\hat{\boldsymbol{\Phi}} = \boldsymbol{V}_p$, where $\boldsymbol{V}_p$ contains the p eigenvectors with largest eigenvalues of the empirical covariance matrix $\boldsymbol{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^\intercal$. Note that this is true irrespective of the average $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$, but in PCA we typically assume that this average is zero. Indeed, if that is not the case, we can easily obtain a smaller error by "centering" the dataset prior to analysis. The optimal low-dimensional encoding (that is to say, the expansion coefficients in the basis $(\boldsymbol{\phi}_j)_{j=1}^p$) of the data is given by $\hat{\boldsymbol{w}}_i = \boldsymbol{\Phi}^\intercal \boldsymbol{x}_i$, which gives an orthogonal projection of the data onto the column space spanned by the eigenvectors. In the following we denote the eigenvalues of $\boldsymbol{\Sigma}$ by $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d > 0$.

- Let σ^2 be the empirical variance of the eigenvalues, $\sigma^2 = \frac{1}{d}\sum_j^d (\lambda_j - \bar{\lambda})^2$ and $\bar{\lambda} = \frac{1}{n}\sum_j^d \lambda_j$. Explain (in words) why σ^2 is a good measure of whether or not PCA is useful to analyse this data.

- For $p < d$, show that
$$L(\hat{\boldsymbol{\Phi}}, \hat{\boldsymbol{W}}) = \sum_{j=p+1}^{d} \lambda_j.$$
Hint: partition the sum $\sum_{j=1}^{d} \lambda_j$ into $\sum_{j=1}^{p} \lambda_j$ and $\sum_{j=p+1}^{d} \lambda_j$ and consider the Pythagorean theorem.

- Let $\boldsymbol{v}_k$ be the eigenvector with k-th largest eigenvalue of $\boldsymbol{\Sigma}$. We define $\bar{\boldsymbol{x}}_i$ as the orthogonal projection of $\boldsymbol{x}_i$ onto the space orthogonal to the first eigenvector $\boldsymbol{v}_1$:
$$\bar{\boldsymbol{x}}_i = (\boldsymbol{I} - \boldsymbol{v}_1 \boldsymbol{v}_1^\intercal)\, \boldsymbol{x}_i,$$
where $\boldsymbol{I}$ is the identity matrix. Let the "deflated" data matrix be $\bar{\boldsymbol{D}} = [\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_2, \cdots, \bar{\boldsymbol{x}}_n]$. Show that the first principal component ($p = 1$) of the deflated data $\bar{\boldsymbol{D}}$ is $\boldsymbol{v}_2$.

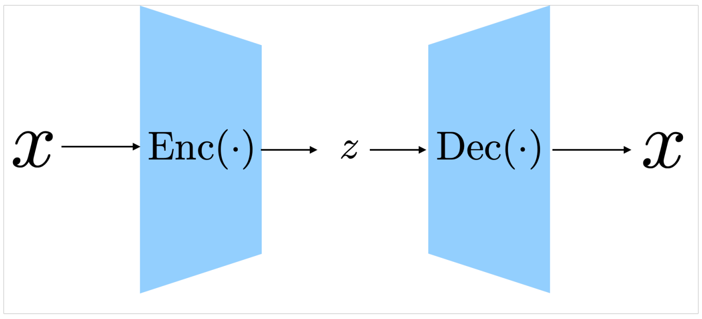

Figure 1: Autoencoder Architecture

Coding

Exercise 2 (Autoencoders — 2 points)**.**

Autoencoders are a popular type of neural networks used to learn compressed representations of data in an unsupervised manner. Autoencoders have two components: an encoder to obtain the latent representation, which is typically of low dimension compared to its input, and a decoder to recover the input using the latent representation. The network architecture of an encoder is given in Figure .1. To train an autoencoder, the target output is the input itself. Thus, you can train it using only the input data. The network is trained by minimizing the following problem:

$$\min_{\theta_E,\theta_D} \mathbb{E}_{x\sim p_x}\left(\|\mathrm{Dec}_{\theta_D}(\mathrm{Enc}_{\theta_E}(x)) - x\|_2^2\right)$$

The goal of this assignment is to train an autoencoder on a simple faces dataset (`https://cs.nyu.edu/~roweis/data.html`) and generate new faces from the network. We have provided you with a starter code that loads the dataset and a simple autoencoder model in the file `latent.ipynb`. The autoencoder model script is in `./models/autoencoder.py`; you can take a look at it to understand the required outputs and inputs structure. Your task in this exercise is to complete the different parts of the successful training of an autoencoder:

1. **Training:** Use a dataloader to obtain the data in batches and train the model for 1000 epochs with a learning rate of 0.001. You can use the Adam optimizer and the Mean Squared Error (`torch.nn.MSELoss()`) loss function. Finally, plot one of the reconstructed images of your choice next to the corresponding true input image in the dataset.

   Hint:

   (a) The optimization can go wrong, and the network might learn to always output the same image, whatever the input. Make sure this is not the case.

   (b) You need to reshape the output from the network.

2. **Latent space visualization:** Compute the latent representation for all the data points and plot it as a 2D scatter plot. Then, choose two points in the dataset that are sufficiently far apart in the latent representation and plot the corresponding images. Highlight these two points in your scatter plot. Ideally, these two images are noticeably different.

   Note: The network is hardcoded to output a 2D latent vector per image.

3. **Latent space interpolation:** Sample points uniformly along the line segment between the two points in the latent space. Highlight this line segment on the scatter plot. You can choose 10 to 100 points along the line segment. Obtain the corresponding images using the decoder function in the autoencoder and plot them using subplots. Choose 10 images to plot from your interpolation. We have provided you with a sample scatter plot and the corresponding interpolation results in Figure .2 and Figure .3.

   Hint: You should observe a smooth transition from one face to the other.

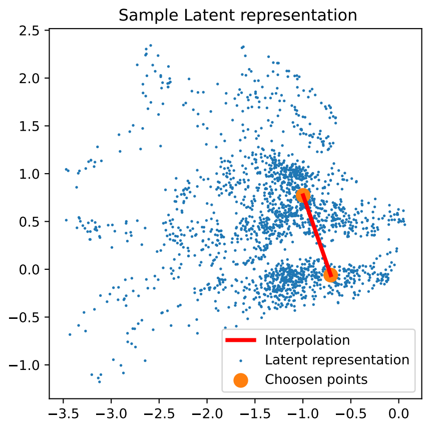

Figure 2: Sample Latent Represenation

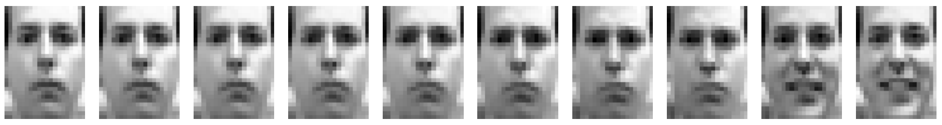

Figure 3: Sample interpolation

4. **PCA:** Reproduce the above steps by replacing the autoencoder with Principal Component Analysis to obtain the 2-dimensional latent vectors. You can use the PCA implementation from `sklearn.decomposition`.

You should add your code and figures in the notebook `latent.ipynb`. accordingly and then submit it. Make sure to run all the cells in the notebook before submitting, ensure that the results are visible, and that the cells follow the correct ordering.

Exercise 3 (Segmentation — 2 points)**.**

Pixel-wise image segmentation is an important computer vision task that involves dividing an image into different segments representing various objects or regions of interest. Such a method can aid in medical diagnostic scenarios where experts can sift through large amounts of data quickly or assist in the automatic diagnosis of several conditions. In this exercise, we will focus on segmenting polyps used in gastrointestinal disease detection. We will use a popular dataset, Kvasir SEG (`https://datasets.simula.no/kvasir-seg/`) [1] to train and test segmentation networks. The segmentation network $f_\theta(\cdot)$ simply takes an image x as input and predicts the segmentation map y. To train such types of networks, we need to ensure that the predicted output $f_\theta(x)$ is as close to the true output y as possible. Closeness can be defined in several ways, and one such method is Binary Cross Entropy (BCE) loss, commonly used in binary classification problems. Binary cross entorpy loss is defined as:

$$BCE(y,p) = \frac{1}{N^2}\left(\sum_{i,j} y[i,j]\log(p[i,j]) + (1-y[i,j])log(1-p(i,j))\right)$$

where $p = f_\theta(x)$. Additionally, several metrics have been developed that are effective for segmentation problems, with one of the recent popular ones being the Dice loss [2], which is defined as

$$DiceLoss(y,p) = 1 - \frac{2\sum_{i,j} y[i,j]p[i,j]}{\sum_{i,j}(y[i,j]+p[i,j]) + \epsilon},\ \epsilon > 0$$

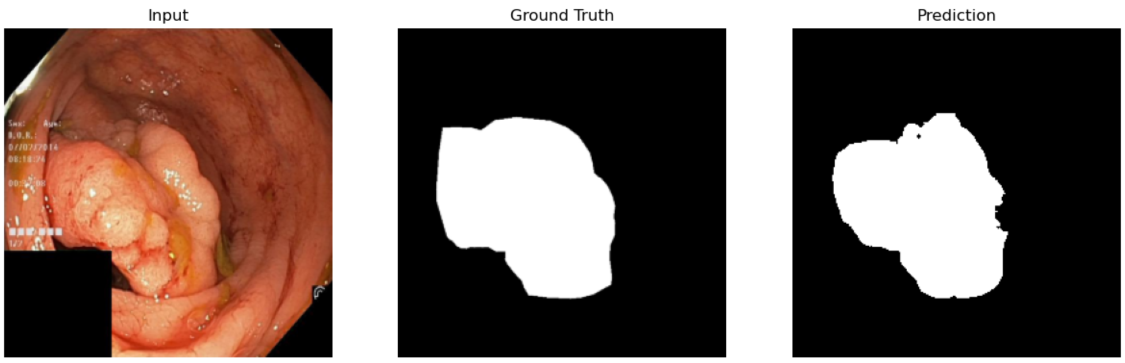

Figure 4: Segementation and prediction for a sample image in the dataset.

To train the network in the assignment the two losses are combined as follows:

$$\min_{\theta} \mathbb{E}_{y,x\sim p_{x,y}} \left(BCE(y, f_\theta(x)) + \lambda_{DiceLoss} DiceLoss(y, f_\theta(x)) \right)$$

where $\lambda_{DiceLoss} \geq 0$ is a user-defined parameter. A sample image and its segmentation are given in Figure .4. The quality of segmentation is typically evaluated using the Intersection over Union (IoU) metric. We have provided the function to compute this metric.

In this exercise, you will test two types of networks: U-Net [3], a popular network that has been successfully used for several biomedical applications, and a simple Vision Transformer (ViT) [4], which employs a transformer-like architecture for computer vision tasks. We have provided you with a simple ViT architecture in the file `./models/vit.py` and the U-Net architecture in the file `./models/unet.py`. We have also provided you with a notebook `segment.ipynb` where the necessary libraries and data are loaded. Your task in this assignment is:

1. Split the data into train, test, and validation sets. The validation set is used to tune the hyperparameters of the network, and the test set is used to report the results.

2. Choose an appropriate batch size and initialize the data loader for each set. Hint: use `TensorDataset`.

3. Define the U-Net model (`UNet(3, 1).to(device)`), the loss, and the optimizer. We have provided you with the following parameters; you can choose to vary them:

   (a) `EPOCHS`: number of epochs trained ,

   (b) `LR`: learning rate,

   (c) `LAMBDA_DICE`: the weight $\lambda_{DiceLoss}$ in the loss function .

   Use `torch.nn.BCELoss()` for BCE loss and `smp.losses.DiceLoss('binary')` for dice loss. You can use the Adam optimizer or experiment with other optimizers. Note that the loss function is defined for a single image; however, you can use it for multiple images at once.

4. Complete the training loop of the code. Evaluate the network's performance using the validation set in terms of the IoU metric using the provided function. You can choose how often you want to evaluate your network on the validation set. Plot the loss curves.

5. Evaluate the model on the test set and report the IoU metric. Plot a few examples from your test set and the corresponding predictions. Note that the plots should show results from several cases, including examples where it worked well, cases where it missed the segmentation completely, and cases where the network partially recovered the segmentations.

6. Report on the hyperparameters used and observations on the test set in the cell `Observations and Results`.
7. Report on the possible reasons for using binary cross-entropy loss for training the model.
8. Repeat the above steps using the vision transformer. We have provided you with a simple vision transformer model in the class `SegmentViT`. Complete the steps again and note down your observations and results in the cell `ViT Observations and Results` and compare the results with those of U-Net.

Submit the notebook `segment.ipynb` with results and code. Make sure to run all the cells in the notebook before submitting; ensure that the results are visible and that the cells follow the correct ordering.

Grading

The coding assignment will be graded based on the code, results, and your observations. We will check the following:

1. Ensure all the steps to train a network are followed. This includes initializing the optimizer, clearing out the gradients, etc.
2. Correctly use the train, test, and valid datasets for training and reporting the results.
3. Visible results: make sure that the notebook contains the plots, not just the code. If the plots are not visible in the notebook, we assume the results are not present.
4. Clearly mention the observations in your experiments. What parameters worked and what didn't. You don't need to run an exhaustive set of hyperparameters. Make sure to choose a few of them and test it.
5. You can split the cells in the notebook if required, but make sure that the entire notebook is in a proper order and coherent.

References

[1] Debesh Jha, Pia H. Smedsrud, Michael A. Riegler, Pål Halvorsen, Thomas de Lange, Dag Johansen, and Håvard D. Johansen, *Kvasir-seg: A segmented polyp dataset*, in *MultiMedia Modeling: 26th International Conference, MMM 2020, Daejeon, South Korea, January 5–8, 2020, Proceedings, Part II 26*, pages=451–462, year=2020, organization=Springer.

[2] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. Jorge Cardoso, *Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations*, in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pp. 240–248, 2017, Springer.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-net: Convolutional networks for biomedical image segmentation*, in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages=234–241, year=2015, organization=Springer

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and others, *An image is worth 16x16 words: Transformers for image recognition at scale*, *arXiv preprint arXiv:2010.11929*, year=2020.