Exercise 3 - Self-Supervised & Unsupervised Learning

67912 - Advanced Course in Machine Learning

Last Updated: 4.7.2024

Due Date: 25.7.2024

1 Exercise Overview

Self-supervised learning (SSL) has been shown to be highly effective for many downstream tasks. In this exercise you will implement a state-of-the-art SSL method. You will then experiment with several downstream applications of self-supervised learning, and explore its usefulness.

First, you will train the VICReg [1] algorithm, which you will build from scratch by yourself, on the CIFAR10 [2] dataset. Second, you will test VICReg's effectiveness using several downstream tasks: 1) Classification by linear probing: how much does self-supervised pre-training contributes to classification? 2) Retrieval: Which samples are similar to one another? 3) Anomaly Detection: Finding anomalies without any explicit supervision. 4) Clustering: dividing the dataset into groups.

You will submit a PDF report of your work (as well as **all** of your code). In this PDF you will answer the questions, and present the requested findings described in Sec. 3 & 4. In the report please explain the difficulties you encountered, what worked and what didn't. If you have additional interesting findings you are welcome share them as well.

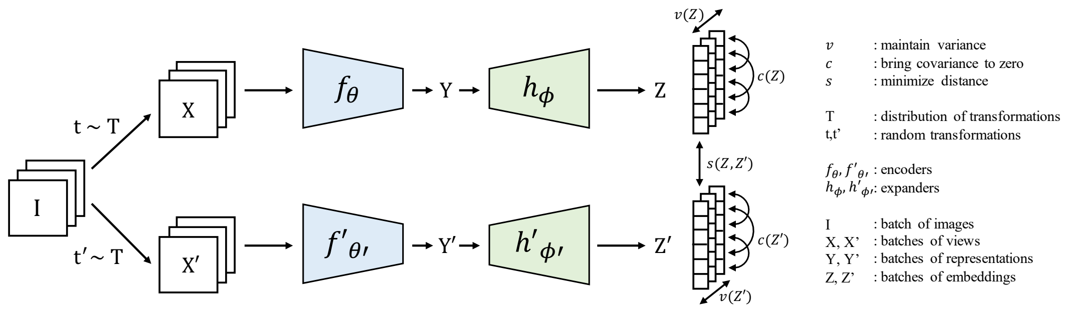
I
t ~ T
t' ~ T
X
X'
f_θ
$f'_{\theta'}$
Y
Y'
h_ϕ
$h'_{\phi'}$
Z
Z'
$v(Z)$
$c(Z)$
$s(Z,Z')$
$c(Z')$
$v(Z')$
v : maintain variance
c : bring covariance to zero
s : minimize distance
T : distribution of transformations
t,t' : random transformations
$f_\theta, f'_{\theta'}$: encoders
$h_\phi, h'_{\phi'}$: expanders
I : batch of images
X, X' : batches of views
Y, Y' : batches of representations
Z, Z' : batches of embeddings

Figure 1: **VICReg's Architecture**

2 Background

2.1 VICReg

2.1.1 Overview

VICReg [1] is a self-supervised method, it relies on: i) an unlabelled training set and ii) some priors on the data, to learn meaningful representations. Specifically, it is very effective for images, due to prior knowledge on image augmentations, which serve as the "priors" made on the data. The purpose of self-supervised methods is to learn a meaningful representation, one that can be leveraged for further downstream applications. VICReg is composed of an encoder and a projection head, which is omitted at test time. In each step, the input image is randomly augmented twice to create 2 views of the same image. Then, a representation and projection is extracted using the encoder and projection accordingly, for each view. Finally, 3 separate objectives are used: 1) *Invariance* of the representation, between the projections of the 2 augmented views. 2) Increasing the *Variance* of each dimension in the projection. 3) Reducing the off-diagonal *Covariance* between different dimensions (variables) in the projection.

2.1.2 Detailed Method Description

Architecture and Notations. VICReg operates in the following way. It trains an encoder f, which is responsible for the extraction of the representation, y. It also trains a projection head named h, which takes y as input and returns a projection denoted as z. **This projection head is used solely for training the encoder, and is omitted at test time**. The encoder is usually a visual backbone (e.g. ResNet50, ViT, etc.). The projection head is typically composed of a few $(3-5)$ Fully-Connected (FC) layers (separated by non-linear operations). VICReg is operated as a "twin tower": In each step, each image (I) is augmented twice. The augmentation function is stochastic, and named T. Meaning, after computing $T(I)$ for 2 separate times, we get two different inputs to our encoder [1]. We denote the 2 different augmented views as x and x'. Then, the encoder (f) is operated on x and x', outputting y and y' accordingly. Finally, the projection head (h) is operated on top of y and y' to receive z and z' accordingly. Formally,

$$z = h(f(T(I))) \quad ; \quad z' = h(f(T(I))) \tag{1}$$

where z, z' are different as the augmentation T is a stochastic function. We denote the projection of each batch B as $Z = z_1, ..., z_{|B|}$ and $Z' = {z'}_1, ..., {z'}_{|B|}$. In addition, for a projection z, we denote each value at each of its d dimensions as $z^1, ..., z^d$. This means the value at the j^{th} dimension of the i^{th} image in a batch, will be denoted as z_i^j.

Fig.1 shows a full visual illustration of VICReg.

Objectives. VICReg optimizes 3 different objectives using the projections z and z' of each image in its batch. We name the 3 objectives as the *Invariance*, *Variance* and *Covariance* objectives. We shall now describe each objective in detail.

Invariance Objective. This objective drives the encoder to be invariant to the performed augmentations (T). E.g.,

[1] Originally, VICReg has 2 separate encoders and 2 separate projection heads. These create 2 different projections, one for each *different* view of each image. In practice (as in our case), it is often implemented with shared weights between the encoders and projection heads, meaning it only has a single encoder and a single projection head. Hence, the only difference left is the augmentations performed.

if one of the augmentations is horizontal flipping, the encoder should output the same representation for an image and its horizontally flipped augmented view. Formally, this objective minimizes the Mean Squared Error (MSE) between z & z':

$$\mathcal{L}_{inv}(Z,Z') = \frac{1}{|B|}\sum_{i=1}^{|B|} MSE(z_i, z'_i) \tag{2}$$

Variance Objective. The variance objective forces each embedding vector (projection) in the batch to be different, by making sure each dimension in the representation is meaningful. This is performed by optimizing the standard deviation of each dimension to be above a known threshold, using the hinge loss. Formally,

$$\mathcal{L}_{var}(Z) = \frac{1}{d}\sum_{j=1}^{d} Hinge(\gamma, \sigma_j) = \frac{1}{d}\sum_{j=1}^{d} max(0, \gamma - \sigma_j) \quad ; \quad \sigma_j = \sqrt{Var(z_1^j, ..., z_{|B|}^j) + \epsilon} \tag{3}$$

Where γ, ϵ are hyper-parameters (typically $\gamma = 1, \epsilon = 10^{-4}$). Note that this objective is operated twice in each step, on both branches of the "twin tower", i.e. with Z and with Z'.

Covariance Objective. This objective aims to decorrelate the variables of the embedding, thereby preventing it to collapse to only a few dimensions. It does so, by minimizing the covariance between the different dimensions of the embedding. The covariance matrix is defined as follows:

$$C(Z) = \frac{1}{|B|-1}\sum_{i=1}^{|B|}(z_i - \overline{z})(z_i - \overline{z})^T \quad ; \quad \overline{z} = \frac{1}{|B|}\sum_{i=1}^{|B|} z_i \tag{4}$$

Hence, the covariance objective is:

$$\mathcal{L}_{cov}(Z) = \frac{1}{d}\sum_{i \neq j}^{d}[C(Z)]_{i,j}^2 \tag{5}$$

Similarly to the variance objective, the covariance objective is also operated on both branches of the "twin tower".

Combines together, these 3 objective functions compose the full VICReg loss:

$$\mathcal{L}_{VICReg}(Z,Z') = \lambda \cdot \mathcal{L}_{inv}(Z,Z') + \mu \cdot [\mathcal{L}_{var}(Z) + \mathcal{L}_{var}(Z')] + \nu \cdot [\mathcal{L}_{cov}(Z) + \mathcal{L}_{cov}(Z')] \tag{6}$$

Where λ, μ, ν are hyper-parameters controlling the weighting of the different losses. Notice[2]: Both the variance and covariance objectives are performed on the projection set $z_1, ..., z_{|B|}$, and **additionally** on the set $z'_1, ..., z'_{|B|}$.

(Highly) Recommended Hyper-parameters. VICReg can be highly sensitive to different choices of hyper-parameters. We give you a list of the hyper-parameters you should use for your VICReg implementation: (Some of them detailed by code in the additional files).

- γ : 1.
- ϵ : 10^{-4}

[2]While this seem redundant in our setting, it is an essential part of VICReg. This is what enables VICReg to train on different modalities seamlessly.

- λ : 25.
- μ : 25.
- ν : 1.
- Projection Dimension (d) : 512
- Projection Head Arch.: detailed in the additional files.
- Encoder Dimension: 128
- $|B|$: 256
- T : detailed in the additional files.
- Learning Rate: $3 \cdot 10^{-4}$
- Optimizer: $torch.optim.Adam(params, lr = 3 \cdot 10^{-4}, betas = (0.9, 0.999), weight\_decay = 10^{-6})$
- Number of Epochs: At least 10 (although ≥ 30 yield much better results).
- Backbone (f): detailed in the additional files.

Algorithm 1 VICReg Training

```
Input: A dataset of images, $I_1, ..., I_N$
for ep in range($\#_{Epochs}$) do
    for $B$ in range($Batches$) do
        $Z = h(f(T(B)))$                                  ▷ $Z = z_1, ..., z_{|B|}$
        $Z' = h(f(T(B)))$                                 ▷ $Z \neq Z'$
        $loss = \mathcal{L}_{VICReg}(Z, Z')$              ▷ Eq. 6
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    end for
end for
return $f$
```

2.2 Probing Pre-Trained Networks for Classification

One dominant application of representation learning is classification. An easy way of utilizing a pre-trained Neural Network (NN) for classification is probing. Probing takes a pre-trained NN and a small labelled dataset as input. It then freezes the pre-trained network, and trains a few FC layers (separated by non-linear operations) on top of the pre-trained NN representations in a supervised manner. The linear layer is trained like any other NN (with SGD), and the objective is the standard classification objective: Cross-Entropy. The most popular setting of probing is linear probing, meaning only a single FC layer is trained (without any non-linear operation). In this exercise you will experiment shortly with probing and analyze its effectiveness in different scenarios.

2.3 Anomaly Detection using Pre-Trained Networks

Anomaly Detection [3] is an important task in machine learning. The setting is simple: for training we are given an unsupervised dataset, containing normal images only. At test time, we will receive images which can be either normal or anomalous (in many ways). The task is binary classification of the test set, 1 for anomalous samples and 0 for normal ones.

The main challenge of anomaly detection is that anomalies can occur in many ways, and a reason can be found to denote any sample as anomalous. Recent research uncovered that the natural bias of pre-trained NN is very effective for certain types of anomalies. In this exercise, you will have to implement such an anomaly detection method. The approach is simple: using a pre-trained NN, we project the images to an embedding space. In this space, and for each test image, we will perform density estimation of the samples using the kNN algorithm and the train set. The higher the density around a sample, the less likely it to be an anomaly. Therefore we will classify samples in high density areas as normal (0), and samples in low density areas as anomalous (1). We do that by setting a threshold for the probability density. In practice, it is more convenient to compute the inverse density score, meaning any samples with: (i) higher inverse density score than the threshold are classified as anomalies - 1 (ii) lower inverse density score than the threshold are classified as normal - 0.

Density Estimation with kNN. Performing density estimation using kNN is almost trivial. In practice since we label anomalies as 1 and normal data as 0, we will compute the inverse density score. Given a test sample, we look at its k nearest neighbors from the train set. We compute the average ℓ_2 distance between the samples and its k nearest neighbors. Our inverse density score is that value, meaning:

$$Inverse\_Density\_Score = \frac{1}{k}\sum_{i=1}^{k} \ell_2(x, kNN(x)_i) \tag{7}$$

Evaluation. In this exercise we will replace the last thresholding stage of anomaly detection, as we only want to evaluate the algorithm. Instead, we will compute the ROC Curve for each algorithm, over all test samples. The ROC Curve is simply plotting the True Positive Rate (TPR) against the False Positive Rate (FPR), computed for each density threshold of binary classification. It has some interesting properties (e.g. bring monotonically increasing) which we will not delve into in this scope. To measure an anomaly detection algorithm's success, we compute the Area Under the Curve (AUC) of its ROC curve. For anomaly detection, this AUC effectively measures the likelihood of an alerted sample (a sample classified as 1) to truely be an anomaly. For the purposes of the exercies, you will use the implementation of `sklearn` to compute the ROC Curve and its AUC.

2.4 Clustering

Clustering is another unsupervised task which is very useful in machine learning. The goal is to separate the data into groups, without any labels. This label-free setting makes clustering to be a highly ambiguous task, with many valid solutions (essentially, any grouping of the data is a valid solution). Generally, some prior is applied to group the data. In this exercise we will use the K-Means algorithm, which assumes the data fits neatly into K well-separated clusters. In this exercise, we will combine it with the prior of our representation learning. We expect the groups to be similar to the different classes of CIFAR10.

Evaluating clustering algorithms is an especially difficult task, due to the high ambiguity of the task. One suggested approach is the *Silhouette Score* which we will use in this exercise. The Silhouette Score of each point measures the difference between (i) the average distance of each sample to its cluster neighbors (ii) the average distance of each sample to the other nearest cluster neighbors. This value is then divided by the maximum of the two. Finally, the Silhouette Score of a clustering (of an entire dataset) is just an average of the sillouhette scores of all the points. Formally,

$$Silhouette\_Score = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{b_i - a_i}{\max(a_i, b_i)}\right) \tag{8}$$

where,

$$\mathrm{a}_i = \tfrac{1}{cluster\_size(i)-1}\sum_{j\neq i, same\ cluster}\ell_2(i,j) \quad ; \quad b_i = \min_{other\ clusters}\left(\tfrac{1}{cluster\_size(j)}\sum_{j, other\ cluster}\ell_2(i,j)\right)$$

3 VICReg ⇒ Laplacian Eigenmaps (70 pts)

You are required to train a VICReg model over the CIFAR10 dataset, then answer the following questions / assignments. The main challenge of this part of the exercise is truly understanding the logic behind VICReg. You will be required to perform ablation studies (modifications) to different parts of the algorithm, until you will "peal" it back to Laplacian Eigenmaps. You will be required to explain how did the changes affect the performance / representations.

Do not use any external code or library other than `numpy`, `scipy`, `sklearn`, `faiss`, `pytorch`, `torchvision`, `matplotlib`, `tensorboard`, `wandb`, `pandas`, `plotly`, and `tqdm`. You are of course allowd to use all basic "native" python packages, such as `math` or `os`. We recommend using `plotly.graph_objects` or `plotly.express` when plotting a small number of data points, for good practice. These plots are interactive and are better for understanding the results than `matplotlib` ones. Saying that, for plotting larger amounts these interactive plots take a lot of time to compile. In these cases we recommend using `matplotlib`'s plotting.

3.1 Resources

We do not recommend running this part of your exercise on your private computer. Instead, run it using Google Colab's **free tier** GPU. If you run out of time, you can try again with another email (HUJI mail / private).

You absolutely do not need to pay for any resources for the purposes of this exercise!
If you run into a problem with this subject please reach out to us as soon as possible by Moodle / reception hours.

3.2 Question & Assignments

For each experiment in the following questions, save the weights from the final epoch. We will need them for following questions, as well as the rest of the exercise. In the ablations questions be consistent with the choice of hyper-parameters (including number of epochs). We require you to present the followings:

1. **(Q1: Training.)** Train VICReg on the CIFAR10 dataset. Plot the values of each of the 3 objectives (in separate figures) as a function of the training batches. In your figures also include the loss terms values on the test set,

computed once every epoch.

2. **(Q2: PCA vs. T-SNE Visualizations.)** Compute the representations of each test image using your trained encoder. Map (using the `sklearn` library) the representation to a 2D space using: (i) PCA (ii) T-SNE [4]. Plot the T-SNE and the PCA 2D representations, colored by their classes. Look at both visualizations (PCA vs. T-SNE), which one seems more effective for visualizations to you? Look at the T-SNE visualization. Did VICReg managed to capture the class information accurately? Which classes seem entangled to you? Explain.

3. **(Q3: Linear Probing.)** Perform a linear probing (single FC layer) to the encoder's representation. Train this classifier on the representations of the CIFAR10 **train** set. Remember to freeze the encoder, i.e. do not update it. Compute the probing's accuracy on the test set. What is the accuracy you reach with your classifier?
Note: classifier accuracy should be at least 60% on the test set.

4. **(Q4: Ablation 1 - No Variance Term.)** Modify the optimized objective, by removing the variance objective term. Using the representations from the modified encoder, perform the same T-SNE visualization from Q2, and the linear probing from Q3 (and include them in your report). Is the new accuracy better or worse? Can you see anything different in the representations visualization? Try to explain the difference in the accuracy using the visualizations.

5. **(Q5: Ablation 2 - No Generated Neighbors.)** Go back to our original VICReg algorithm, with the encoder. This time we will remove the generated neighbors from the method: First, compute the representations of your original VICReg, on all of the training set. In each step of training and for each image in the batch, use these representations to find the top 3 nearest neighbors, and randomly select 1 of them. Use these images as your other second view for the VICReg algorithm. 2 Practical Tips: (i) We find that training this algorithm for only a single epoch is more beneficial. (ii) We recommend you to compute the neighboring indices of each image in advance, and delete the original VICReg model from your (GPU) memory. This will save both run time and GPU space.
Train this method for the same number of epochs as the VICReg. Compute the linear probing accuracy, and report it. Is the accuracy different from the original linear probing from Q3?

    - If no, explain what added value do you think the generated neighbors adds to the algorithm.
    - If yes, explain why do you think this change had no effect (what compensates the things that are missing?).

6. **(Q6: Ablation 3 - Laplacian Eigenmaps.)** After removing the generated neighbors and the amortization alone (well... we wanted to), we would like to remove them both at once. To do so, we will perform Laplacian Eigenmaps [5] representation learning on the training data of CIFAR10, using the `sklearn`'s implementation. Unfortunately, an ℓ_2 distance on the image pixels is not very effective, and is very costly on resources. Hence, we will need to perform some primitive dimension reduction before that. To take into consideration the spatial structure of images, we will use an untrained small CNN, built from 4 CNN layers (no non-linear operations needed), with a kernel of size 5, latent dimension of 32, and no padding. After that project the output to a 1024 size vector. Important implementation details:

    - Use `eigen_solver="arpack"`, `n_jobs=10`.
    - The runtimes of this algorithm are too slow. Use only 10K samples instead of the whole training set. For a fair comparison, train VICReg with 10K samples as well for at least **50** epochs (same number of batch iterations).

Report this VICReg's version linear probing results (should be close to the original results). Remember to use only 10K samples for the probing's training data as well.

- Since this algorithm is pretty slow, we recommend to save the embeddings you extracted including their labels. We will need those for the Clustering section of the exercise.

Since Laplacian Eigenmaps uses Latent Optimization (LO), we will test it as follows: Take 90% of the trained codes, and use them as training data for linear probing. Use the rest 10% of codes as the test data for the linear probing. Remember to use a stratified split according to the labels.
Compute the linear probing accuracy. Compare this to VICReg's linear probing accuracy. Plot the embeddings using T-SNE as in Q2. Compare this result to the visualization of VICReg. Based on this visual and linear probing comparison, which method (VICReg vs. Laplacian Eigenmaps) is more effective for downstream object classification? Explain your answer.

7. **(Q7: Retrieval Evaluation.)** Now that we slowly "pealed" VICReg back to the laplacian eigenmaps algorithm, we wish to evaluate it qualitatively. For each method from **Q1** (VICReg), **Q5** (No Generated Neighbors) and **Q6** (Laplacian Eigenmaps) perform a qualitative retrieval evaluation. That means:

   - Select 10 random images from the training set, one from each class.
   - For each selected image, use the representations of each of the evaluated methods, to find its 5 nearest neighbors in the dataset.
   - Plot the images together with their neighbors.
   - Using the same images, perform the same visualization for the 5 most distant images in the dataset.

   Using this visualization, explain what attributes each method attends to. What are the differences you see between the different methods? Which one excels at keeping close images together? Which one excels at keeping distant images far apart? Discuss the differences between the methods, as seen by this visualization. You may select more than 1 image for a specific class if you wish to get a better understanding (Although it is not mandatory).

4 Downstream Applications (20 pts + 10 optional bonus pts)

In this part of the exercise we will use pre-trained weights from VICReg and representations from Laplacian Eigenmaps to perform Anomaly Detection and Clustering. The main challenges of the exercise are (i) understanding the ambiguity in anomaly detection and (ii) **(Bonus)** understanding the struggle of evaluating clustering algorithms.

4.1 Resources

Once you have extracted the pre-trained representations for the training and test sets (Q1), it should not be a problem to run the rest this part of your exercise on your private computer.

4.2 Anomaly Detection (20 pts)

Setting Definition For evaluating anomaly detection, we need to define our normal data and anomalies. We will use the CIFAR10 dataset as our normal data, meaning the experiments should be performed using your **already trained** VICReg algorithm. Our anomalies will be images from the MNIST dataset. This means the training data is the CIFAR10 training data, but the test data is the combination of the CIFAR10 test data and the MNIST test data. All images from CIFAR10 should be labelled 0 (normal), and all images from the MNIST dataset should be labelled 1 (anomalies).

Please perform the described experiments and discuss their results:

1. **(Q1 - Anomaly Detection.)** Using the CIFAR10 training data as reference for normal data, compute the $k - NN$ density estimation for all the (CIFAR10 + MNIST) test set representations. Do this for both (i) VICReg (ii) VICReg without generated neighbors (Sec. 3, Q5) . Use $k = 2$.
2. **(Q3 - ROC AUC Evaluation.)** Plot the ROC Curve of both methods. Use the `sklearn` library for creating these figures. In the title / legend incorporate the AUC of each method. Which method is 'better'? In a sentence or two, explain why do you think its better.
3. **(Q3 - Qualitative Evaluation - Ambiguity of Anomaly Detection.)** Plot the 7 most anomalous samples according to VICReg and Laplacian Eigenmaps, in two separate rows (you can split to different plots if more convenient). Look at the results. Explain what aspects each method found to be anomalous. Keeping in mind we did not give either of the methods any clues regarding which anomalies are we looking for, do you still think one is better than the other? Explain your answer.

4.3 Bonus: Clustering (10 pts)

This part of the exercise (Clustering) is not mandatory!

We will now go back to the original CIFAR10. Use your pre-trained parameters for VICReg and Laplacian Eigenmaps from Sec. 3 Q1 & Q6 for answering the following questions:

1. **(Q1 - Clustering using K-Means).** Use the implementation of `sklearn` to cluster the CIFAR10 training set to 10 clusters. Do this using the representations of VICReg once, and a second time for Laplacian Eigenmaps. For Laplacian Eigenmaps use your $10K$ representations and their labels from Sec. 3, Q6.
2. **(Q2 - Visualizing the Clusters in 2D).** Perform a dimensionality reduction to 2 dimensions using T-SNE [4], for the representations of both methods (separately). For each method, plot the reduced embeddings in a 2D space twice (side by side): (i) First, colored by their cluster index (according to the matching clustering from Q1). (ii) Second, colored by their actual class index. In each figure, also plot the clusters centers in black color.
   Look at the results, which method looks more successful at finding clusters? Which method looks more successful at separating between the classes? Explain your answer, keeping in mind that similarly to Anomaly Detection there are no clues for which clusters to look for.

3. **(Q3 - Quantitative Analysis).** Use the *Silhouette Score* on the clusterings of both methods (in their original embedding dimensions). Report the 2 different scores. Is this coherent with what you see in your visual analysis? Explain your answer.

5 Grading & Requirements

5.1 Ethics & Limitations

This is an advanced course, therefore we will have 0 tolerance for any kind of cheating. We are stating it very clearly that you are forbidden from doing the followings:

- Using any external github repository.
- Sharing any part of your code with other students.
- Using any external library other than `numpy`, `scipy`, `sklearn`, `faiss`, `pytorch`, `torchvision`, `matplotlib`, `tensorboard`, `wandb`, `pandas`, `plotly`, and `tqdm`. You are of course also allowed to use all basic "native" python packages, such as `math` or `os`.

5.1.1 Github Copilot & ChatGPT

We **allow** using automatic tools, such as github copilot and even the infamous ChatGPT, for the exercise. Saying that, we do have a few restrictions for these tools: We require to note and explain (in the submitted code itself) every part of the code that was written by these tools. Also, in your PDF (as a separate section) explain how did you use these tools, and for which aspects of the exercise you found them useful.

5.2 Submission Guidelines

5.2.1 PDF Report

The PDF report should be 14 pages at max. If you decide to do the bonus part (Sec. 4.3) you may have 2 additional pages, for a total of 16 pages. Answer the questions from Sec. 3 & 4 in it, marking clearly where are the answers to each questions. We are note requiring any format. Explain your results when needed and analyze them.

5.2.2 Code

Submit all of your code used to perform the experiments and evaluations of this exercise. Additionally, prepare 3 main files as described below:

Jupyter Notebooks. TL;DR No jupyter notebooks in the submission.
We will not accept any code inside a Jupyter Notebook. Meaning any jupyter notebook submitted will simply be ignored.

It is of course allowed to develop the code for the exercise using notebooks, but for the submission convert the files to standard python files.

5.2.3 Submission

In your submission should be a zip file including the following:

- A README file with your name and and cse username.
- Your code for the VICReg model training, ablations and evaluations.
- Your code for the Anomaly detection part, including visualizations and evaluations.
- Your code for the Clustering part, including visualizations and evaluations.
- A PDF report answering the questions / requests from Sec. 3 & 4.

References

[1] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[3] Tal Reiss, Niv Cohen, Liron Bergman, and Yedid Hoshen. Panda: Adapting pretrained features for anomaly detection and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2806–2814, 2021.

[4] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.