Coding test

Write code in python with either the pytorch or tensorflow libraries in a **single** jupyter notebook to solve the 2 tasks below. Submit the pdf of the already run notebook, so that I do not have to retrain it from scratch and all results are visible.

I will look at how you formulate the task and how you implement the ideas. The evaluation is not about coding style. Also, you are not expected to perform demanding training. Use COLAB if you do not have access to GPUs.

Content of the jupyter notebook:

1. code to define and train the neural networks
2. code to demonstrate the performance of the trained model on the test set
3. visualization of the performance of the trained model on the test set
4. visualization of the **quantitative** performance during training and on the test set
5. text to explain how the tuning and the final performance evaluation are done
6. write comments in the notebook that explain your reasoning and what each function does.

Task # 1

Build and train a denoising autoencoder (with at least 4 encoding/decoding layers) for a dataset of 100 grayscale natural images of size 64×64. During training use uniform noise instead of Gaussian noise. Evaluate the denoising autoencoder on a separate test set and use additional Gaussian noise (instead of uniform noise). Describe your observations.

Task # 2

In this task we train a sender S to build a binary message for a receiver R. The message is the binary order of two concatenated images, *i.e.*, which image was first in the stack and which one was second. Then, R receives the binary message and the two images in random order and has to correctly identify whether that order was the same as the order used in the stack for S. To make things more interesting, we also augment the images before feeding them to R. The data augmentations should be basic ones such as flipping, 90 degrees rotations, color jittering and additional zero-mean Gaussian noise.

To build this system, take 100 images from CIFAR10. For each pair of images x_1 and x_2 from CIFAR10, concatenate them into a single tensor $x = [x_1, x_2]$. Then, train a model S to encode x into a scalar value. Take the thresholded output

$$b(x) = \begin{cases} 1 & \text{if S}(x) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We also write $b(x) = B(\text{S}(x))$, where $B(\cdot)$ denotes binarization. Now map $b(x)$ to a learnable embedding $\xi_{b(x)}$ from a look-up table with only two possible embeddings ξ_0 and ξ_1. Create a new input tensor by combining: 1) the embedding $\xi_{b(x)}$ and the augmented version x'_i of x_i, with $i = 1, 2$, stacked in random order as $\hat{x} = [x'_{1+p}, x'_{2-p}]$, with $p \sim \mathcal{B}(1/2)$ and $\mathcal{B}(1/2)$ denotes the Bernoulli distribution with probability $1/2$. This input is then fed to another model R. We want to train R to output p (*i.e.*, 0 if the order of the tensor fed to S is the same as the order of the tensor $\hat{x}$ fed to R, and 1 otherwise) . The training is written explicitly as

$$\min_{R,S,\xi_0,\xi_1} \sum_{\substack{x_1,x_2 \in \text{Dataset} \\ x=[x_1,x_2], x'_{1,2}=DA(x_{1,2}), \\ p\sim\mathcal{B}(1/2), \hat{x}=[x'_{1+p}, x'_{2-p}]}} \left|\text{R}(\xi_{B(\text{S}(x))}, \hat{x}) - p\right|^2 . \tag{2}$$

The trainable networks are S, R and the embeddings ξ_0 and ξ_1. The design of the architecture is your choice.